TURING 图灵程序设计丛书

MANNING

Sass and Compass IN ACTION

Sass与Compass实战

[美] Wynn Netherland Nathan Weizenbaum Chris Eppstein Brandon Mathis 著

刘炬光 赵锦江 张浩然 译

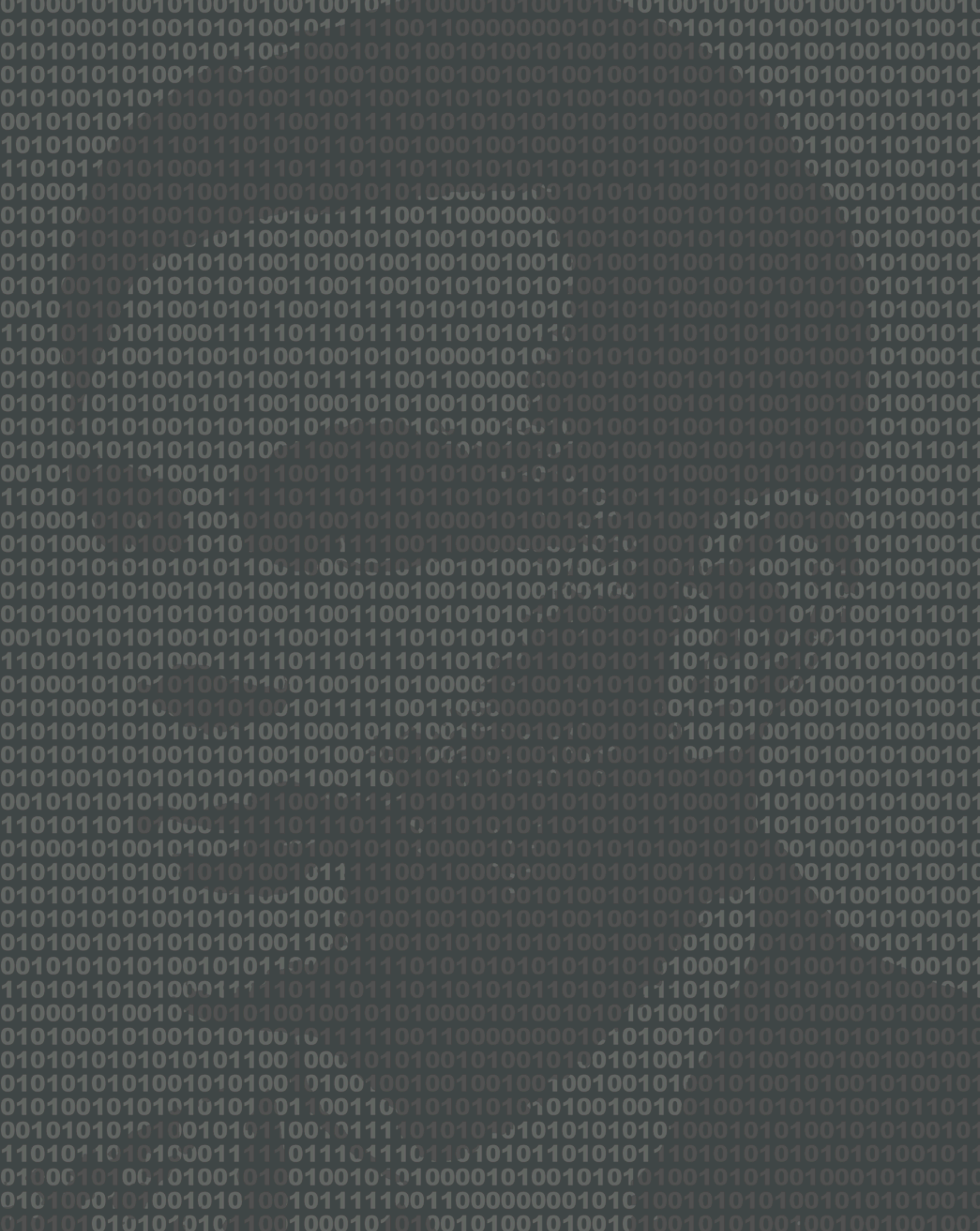


Compass.app & Fire.app开发者 薛良斌（布丁，hlb）

豆瓣前端负责人 张克军

鼎力推荐

人民邮电出版社
POSTS & TELECOM PRESS



作者介绍

Wynn Netherland

有近20年的Web从业经历。他主笔和参与撰写过许多Web开发方面的图书。他喜欢在GitHub上工作，经常在行业会议上演讲，并“出没”于开发者聚会，有时候还会在屋后的门廊里弹吉他。

Nathan Weizenbaum

毕业于华盛顿大学，专业是计算机科学和哲学，他从Sass诞生之初就是首席开发者。目前，他是谷歌的软件工程师，负责Gmail开发。

Chris Eppstein

Compass框架的创建者，Sass核心团队成员。毕业于加州理工学院，软件工程师，有10年以上为硅谷创业公司创建网站和应用程序的经验。痴迷于前端开发，目前在LinkedIn从事前端架构和开发者关系相关的工作。

Brandon Mathis

Compass核心团队成员，基于Jekyll的可扩展博客框架Octopress的作者。他目前是MongoHQ的一名设计师。

TURING 图灵程序设计丛书

Sass and Compass in Action

SASS与Compass实战

[美] Wynn Netherland Nathan Weizenbaum
Chris Eppstein Brandon Mathis 著

刘炬光 赵锦江 张浩然 译

人民邮电出版社
北 京

图书在版编目（CIP）数据

Sass与Compass实战 / (美) 尼德兰
(Netherland,W.) 等著 ; 刘炬光, 赵锦江, 张浩然译.
-- 北京 : 人民邮电出版社, 2014.6
（图灵程序设计丛书）
ISBN 978-7-115-35301-6

Ⅰ. ①S… Ⅱ. ①尼… ②刘… ③赵… ④张… Ⅲ. ①
网页制作工具—程序设计 Ⅳ. ①TP393.092

中国版本图书馆CIP数据核字(2014)第068206号

内 容 提 要

本书共分为10章，旨在完整介绍两个工具：Sass和Compass，从而引领读者通过框架高效地构建样式表，创建动态页面。本书介绍了Sass如何通过选择器嵌套和变量来帮助避免重复，以及通过继承和混合器等特性更加高效地重用通用样式，减少重复编写工作。学完本书后，你一定能对Sass和Compass有一个全面的理解。

如果你是Web设计师或者前端开发人员，那本书定能让你受益匪浅。

◆ 著　　[美] Wynn Netherland　Nathan Weizenbaum
　　Chris Eppstein　Brandon Mathis
译　　刘炬光　赵锦江　张浩然
责任编辑　李松峰
执行编辑　李　静　张　庆　杨　琳
责任印制　焦志炜

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路11号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　　印刷

◆ 开本：800×1000　1/16
印张：13
字数：308千字　　2014年6月第1版
印数：1－4 000册　　2014年6月北京第1次印刷
著作权合同登记号　图字：01-2014-2244号

定价：49.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京崇工商广字第0021号

版 权 声 明

献给那些精心打造网页并因他们手中的作品而欣喜不已的Web开发与设计人员。

推荐序一

回顾一下做过的项目，会发现我们曾经多么忽视CSS！JavaScript有着高大上的设计模式和模块规范，分布在精心设计过的目录里，而CSS仅仅被简单地堆积在一个文件里。它不重要吗？前端开发最基本的任务就是还原设计，离不开用CSS实现那些复杂的布局和绚丽的效果（后面还会提到它在移动Web开发中的重要性）。它简单吗？HttpArchive统计全球Top100网站平均单个页面CSS达到31 KB，在频繁的迭代中维护它们并不轻松。更不要说众所周知的痛点——浏览器兼容性。CSS既然如此重要，又不简单，为什么没有像它的小伙伴JavaScript那样得到重视和发展呢？

种种原因中有一个很明显——它不是语言。CSS开发是基于渲染原理和W3C标准（及事实标准）来设计规则的，这里边没有算法，没有数据结构，没有逻辑控制。这让习惯语言开发的程序员无从生成思路。一个好的后端程序员可以很快掌握JavaScript，但面对CSS就会变得茫然无助。于是便会简单粗暴地把所有代码随意写在一起，所有布局都用float实现。表面上看把问题解决了，实际上是在到处"埋雷"，到处补"破窗"。这种慢性后果并没有激发CSS在语言方面更快的发展。保持简单，复杂的效果让Javascript/Flash去干吧，似乎问题就能回避。但很快移动端的崛起打破了秩序，先是把Flash宣判了，JavaScript在手机上的性能和能耗成为瓶颈，CSS3则在GPU的作用下看起来异常完美，于是一下子被推上风口浪尖。曾经那些尘封的草案又被唤醒。各大浏览器厂商竞相实现最新特征。炫酷的2D/3D动画效果、高端的FlexBox布局……CSS3一夜之间上了头条。随着应用的深入，最初的快感很快退去，新的兼容性问题又来了，浏览器的种种bug，闹心的厂商前缀，刚学到就被废弃的标准……在这种形势下CSS的开发想保持简单也难。

浮华过后，人们开始认真思考如何提高CSS的开发效率和可维护性等工程问题。Sass早在2007年诞生于"工具控"的Ruby社区，开始服务的对象自然是Ruby工程师，是为了让他们能像写Ruby那样写CSS。直到推出SCSS这种语法风格，Sass变得更像是CSS的扩展和增强，可以跟原生CSS混写，很快被前端工程师接纳了。现在在我的项目里，一眼望去都是.scss文件。据我所知，国内各大互联网公司前端团队的CSS开发也都转向Sass或LESS。如果你是前端工程师或有志成为前端工程师，非常有必要把这块纳入到自己的技能体系中。

Sass让CSS变成语言，可以像其他语言那样去组织模块、封装、复用代码，让CSS开发看上去非常有"设计感"。在大规模的网站开发中，Sass的用法变得至关重要。其实任何工具都是双刃剑，用好事半功倍，反之则有破坏性。比如，写Sass很容易就写出多层嵌套，生成的CSS选择器就会有很多级，不仅低效，维护上也很头疼，以至于出现BEM这种用法。在代码设计上，继承和混合器有什么区别，怎么抽象和封装更好，等等这些由用法引申出来问题值得在实践中潜心思考。

在技术社区里经常能看到各种文章介绍Sass的用法或观念或技巧，辨清优劣，吸收精华，前提是要先对它有系统的了解。虽然也能搜到不少中文资料，但很碎片且质量参差不齐。像我这种东看点西看点的人，之前着实走过不少弯路。在一次前端标准化交流会（w3ctech）上，正好碰到李松峰老师，吐槽如今光是引进一些介绍眼下流行的JS库的书，实际上某些JS库虽然流行但在项目中未必适用。反倒像Sass如此普及而且适用面又广的技术，没有一本中文书。没想到松峰老师很快就促成本书的引进，并找到锦江（勾三股四）等有丰富开发经验的人翻译。我对本书的出版充满期待，并相信它一定会把更多人带入CSS的世界。

克军
2014年3月4日

推荐序二

CSS1在1996年提出时，目的是作为一个简单的样式机制，让作者与读者都能为HTML文件添加或改写样式，但是仅限于字体、颜色、框线、间距等。在当初的建议书①的附录有这句话：

我们不指望CSS会进化成一门程序语言。

这句话在2014年读来，是多么地讽刺。今天如果你不掌握任何一种CSS预处理器，大概都不好意思说自己是个合格的前端工程师。

当时规划CSS语言的人们认为，每个人都应该可以轻松掌握CSS，所以它不能太复杂，更不应该有一般程序语言具备的变量、逻辑、函数，等等。但是随着现实需求越来越多，CSS陆续增添了排版布局、阴影圆角、动画分镜，甚至3D特性，使得CSS网页设计早已不是一件简单的事情。更何况我们还得面对"一千零一个"浏览器臭虫与兼容问题，以及像是罚写小学生的众多厂商前缀。撰写CSS成了一种黑魔术，更别提要维护它了。

幸好我们不需要绝望，随着LESS、Sass、Stylus等CSS预处理器的兴起，我们有了另一种选择。网页设计师与开发者可以"别做重复劳动"（DRY，Don't Repeat Yourself），撰写CSS也仍然可以充满乐趣。

Sass 3.0版在2010年5月释出，增添了与CSS兼容的SCSS语法，并且搭配Compass的强大社群支持，从此迈向了主流道路。笔者在2013年底参加了CSSDevConf，开场Keynote的知名设计师Zoe Gillenwater更建议所有网页设计师都应该开始学习Sass。会场上几乎没有人讲"CSS预处理器"，只问："你的项目用了Sass吗？"时至今日，Chrome与Firefox的开发者工具都已经内建SCSS与LESS语法的支持。

同样是在2010年，当时我仍然在自己创办的设计公司Handlino服务。每次我遇到设计师朋友们，都会兴奋地向他们介绍Sass/Compass。只要我稍稍展示变量与图片精灵（第6章）功能，所有人都立刻想要改用Sass/Compass。但是当他们发现"天啊，这玩意得要会使用命令行"，热情立刻就被浇熄了："你别闹了，我不可能学会的！"

那年年底的一个周末，我的同事tka心血来潮，撰写出了一个跨平台的Sass/Compass图形界面工具，立刻受到大家的欢迎。我们决定将它命名为Compass.app②，将程序开源释出，并且在网络

① http://www.w3.org/TR/REC-CSS1-961217#appendix-e。

② http://compass.kkbox.com/。

上贩卖预先编译好的软件。我们承诺捐出30%获利到Compass指定的慈善机构（Compass是一个慈善软件），因而获得了Compass作者Chris Eppstein（也是本书作者）的支持。截至去年，我们已经累积捐出超过15000美元。以一个开源软件来说，这实在是个不可思议的成绩。

Sass已经释出了最新的3.3版，Compass也即将迈向1.0版这个重大的里程碑。很高兴听到《Sass与Compass实战》即将出版的消息，更感谢李松峰老师与译者群投注的辛苦。相信你也能重拾撰写CSS的乐趣。

薛良斌，Compass.app & Fire.app开发者

2014年3月15日

译者序一

刚刚过去的2013年，被称为中国互联网金融发展的元年，O2O概念火遍大江南北。QQ自然不甘落后，整个2013年我都在负责QQ查找的商业化推进。来自上层的压力以及各方的重视，使QQ查找始终以每周最少一个版本的速度快速迭代。一段时间下来，我们发现CSS变得越来越臃肿，不可维护和冗余代码越来越多。

之前很早就在社区了解到了Sass和Compass的存在，但没有形成意识上的重视。一直以来手写静态CSS的思维习惯，使得自己并没有看到通过框架来动态创建样式的巨大优势，甚至认为Sass和Compass是非常呆笨的，是给不会CSS的人使用的。但面对日益庞大的CSS代码，寻找一个有效的技术解决方案就变得非常必要了。

深入之后，才发现自己之前的想法完全错了。Sass并不能让你写出更好的CSS，但是却可以让你更高效地写出高可维护性的CSS。这句话怎么理解呢？ 如果你本身对CSS一无所知，那么Sass并不能帮助你实现从菜鸟到CSS大师的转变，Sass的使用建立在你对CSS有一定了解的基础上，它有着一系列令人激动的特性。全面而详细的特性介绍自然需要读者去详读此书，这里我只挑QQ查找受益颇深的几点分享给大家。

- Sass对原生CSS `@import`指令的扩展，让我们可以将分布在不同文件中的样式最后合并到一起，这样通过把样式分散到多个更小的文件中，整个CSS的结构立马变得更加清晰了。
- 变量机制，使得标准的颜色值、行高、列宽再也不用“东一榔头西一棒槌”地分散各处，有了一个统一的地方归纳整理。设计师哪天想变，只需要修改一处即可，省力又省心。
- 标准样式的抽离，像调用函数一样，轻松将统一的样式抽离出来，避免了复制来、复制去，一旦交互变动， 只需要修改抽离出来的混合器一处。
- 样式继承的概念，使得我们对于查找中各种btn的修饰变得清晰，无论何种btn，统一从父级btn继承基础样式。
- 自动合图实在是太方便了，直接把切图扔到图片目录，对应的合图和CSS就全部自动生成了。

诸如此类的好处实在太多， 留待各位读者通过阅读本书学习和应用吧。

好风凭借力，送我上青云。自从引入了Sass和Compass，QQ查找的样式维护成本大幅下降，给我们带来了莫大的好处。

抱着对Sass和Compass深深的爱与感激，在2013年上海W3CTech的分享大会上，我作为嘉宾重点分享了Sass和Compass，也正是这样一个机会，让我有幸最终结识了李松峰老师。他一直关

注国际前沿技术的最新发展，总希望把最先进最好的东西带给国内的开发者。他非常认可Sass和Compass的发展思路，促成引进了《Sass与Compass实战》这本书。当李老师找到我说想请我翻译这本书的时候，我便一口应允下来，并承诺全力以赴，国内现在太缺少这样一本全面介绍Sass和Compass的书籍了。

感谢李松峰老师对本书提出的修改意见，感谢李静老师对本书最后审定和出版付出的努力。

最后，如果读者是初次接触Sass和Compass，我相信这本书一定可以为你打下非常坚实的语言基础，使你进入这一新的技术领域。即便你已经写了很多年Sass了，本书也一定能够加深你对Sass和Compass高级特性的理解，从而把它们更好地应用到你的项目中。

刘炬光

2014年3月4日

译者序二

样式表是前端工程师几乎每天上班都会碰面打招呼的挚友。而在我看来，它同时也是一项被同行们严重低估的技术。很多人觉得样式表就是CSS，写起来繁琐、枯燥又没有技术含量。但很少有人为此而思考：这是命中注定的吗？撰写样式表就应该是现在这个方式，还仅仅是因为我们习惯了撰写CSS？也许我们的工作可以更简单、更顺畅、效果更好？

说到这里，真的要感谢Sass和Compass，更要感谢Ruby这门神奇的语言以及它神奇的社区力量。正如Ruby的作者松本行弘所说："我们需要从人的角度考虑问题，人们怎样编写程序或者怎样使用机器上的应用程序。我们是主人，它们是仆人。"作为一套基于Ruby实现的优秀工具，Sass和Compass在样式表这个领域里，把这句话体现得淋漓尽致。

通过本书，你将领略到Sass和Compass从变量、混合器等特性，再到补齐CSS3的浏览器厂商前缀、自动化生成精灵图、可配置路径等全方位的优势。本书的讲解浅显易懂，又能逐步解开CSS开发中固有的深层问题，最后让你豁然开朗。

还有一点至关重要，Sass和Compass不仅是开发者工具，而且还是Web设计师的设计工具（或许前后颠倒过来说更合适）。现如今很多专业之间的界限已经越来越模糊了，我个人非常期待国内的前端与设计因为有了这样的工具，而更乐于交融在一起。

对我而言，翻译本书的过程，同时也是重新认真温习Sass和Compass的宝贵机会。记得自己第一次看到Sass时就被深深地吸引，果断去给自己的博客重构了主题。接着又逐渐接触到Compass和几个相关扩展，一边用，一边感叹相见恨晚。后来我把它们用到了更多的地方，顿时优越感十足。

如果你还没有接触过类似Sass的CSS预编译工具，如果你还没有体验过Compass带来的轻松与便利，那么相信看过本书之后，你一定会迫不及待地跟刀耕火种似的CSS工作说再见，全面拥抱Sass和Compass。

最后很高兴能和优秀的图灵出版团队一起工作，并通过译书的方式把优秀的技术和知识传递给更多的人。文字工作者在我眼中一直是非常"高大上"的，能够参与其中，与有荣焉。

谢谢！

赵锦江

2014年3月4日

译者序三

这是一个最好的时代，这是一个最坏的时代。互联网的飞速发展，以及移动设备的推波助澜，为互联网从业者带来了前所未有的机遇。从来没有哪个时代像今天这样，互联网能受到人们的如此青睐，并在人类的生活中扮演如此重要的角色。如果把互联网开发过程比作是一场战役，那么前端工程师和设计师一定就是冲锋陷阵的排头兵，他们永远冲在开发的第一线，他们创造的是用户看得见摸得着的部分，他们的体验决定了用户的体验。然而，随着项目的体积越来越大，用户对产品设计的要求越来越高，样式表的体积越来越大，结构也越来越复杂，每次对样式表进行添加和修改都成了前端工作者的噩梦。

好在我们有了Sass。它基于普通的CSS，并在其中加入了很多令人激动的特性，比如混合器、嵌套，以及函数。Sass完全终结了以往编写样式表时枯燥无味的过程，让每次编写样式表都成了一次充满惊喜的冒险。更重要的是，我们还拥有Compass，它在Sass的基础之上，封装了一系列模块和模板，对Sass的功能进行了强有力的补充。借助Compass，在网上搜索过时的CSS片段并插入到自己的CSS文件这样的做法已经成为过去时。我们可以搜索经过精心设计的Compass扩展，并轻松地将它们加入到我们的项目中，同时能在第一时间得到扩展的更新。Sass让CSS更像是一门语言，它大大提高了CSS代码的可重用性，并让我们能够编写更有效率的CSS代码。Compass为CSS添加了模块性，我们可以像搭积木一样，一块一块地将这些优秀的扩展堆叠起来，轻松创造出不可思议的效果。

Sass和Compass不仅功能强大，学习起来也很轻松。你无需像其他编程语言一样掌握各种数据结构和编程模式，只要熟悉基本语法，掌握简单的逻辑语句，借助强大的内置函数，就可以很快上手并投入实际应用。然而，这并不意味着Sass和Compass就如此简单，即使你已经有了多年的使用经验，依然能从其中发现新的东西。本书就是一本适合于所有开发者的书，无论新手还是老手，你都能在其中找到想要的东西。

最后，要衷心感谢李松峰老师和其他编辑老师，这样一本好书的出版离不开你们的努力和帮助。同时也要感谢另外两位优秀的译者，与你们合作，让我能够努力把事情做得更好。

编写静态的CSS已经成为过去时，创建动态样式表才是未来。少年们，快行动起来，时代在召唤！

张浩然

2014年3月5日

关于本书

我们中的许多人都从社区学习新技术，比如从短篇博文或简短示例中学习样式特殊处理和其他技巧。本书旨在完整地介绍两个工具：Sass和Compass，丰富读者的CSS工具箱，使他们成为更好的CSS编写者。当演示示例程序时，我们将采取系统的方式来讲解Sass的语法，并将相应的模式应用到Compass框架。我们期望在读完本书后，你能对Sass和Compass有一个全面的理解。

本书读者对象

本书主要面向两类读者。第一类是设计师，他们写过很多CSS样式，但可能未及思考如何将多个部分的样式编写过程自动化。第二类是全栈工程师，使他们了解如何在一个项目从开发到最后上线的生命周期中，像处理其他项目资源一样处理样式、图片和字体文件。

路线图

如果你是初次接触Sass和Compass，那么最好先读一读附录A和附录B，其中包括安装指南以及学习本书所需的其他知识。

第1章总揽Sass的强大性能，它不仅有令人激动的特性，还会带给你不同于静态样式枯燥乏味的全新体验。我们还将通过示例引导你尝试使用Compass框架，这些示例应用了Sass的特性构建实用程序。

第2章更深入地介绍Sass，涵盖变量、混合器以及其他一些语言特性。它们是本书其余部分的基础。

第3章专注于探讨应用最为广泛的CSS网格布局。你可以看到，Sass能帮助你轻松达到目的。

第4章重回Compass框架，以更广阔的视角介绍Compass如何减少样式编写过程中的冗余步骤。

第5章总览Compass的CSS3相关模块，并介绍如何通过Compass来对那些常见的CSS3特性实现与厂商无关的实现。

第6章介绍关于CSS精灵的有趣实验，这是每位设计师都应该了解的高级特性。

第7章介绍如何通过Compass的编译功能优化样式，以进行开发调试和发布部署。以此为基础，第8章介绍如何在部署时压缩和减小样式体积的高级技巧。

第9章介绍Sass的高级脚本技术，高级开发者将受益匪浅。第10章扩展了这一主题，告诉你如何编写自己的Compass插件。

代码约定和下载

代码清单和正文中的源代码使用等宽字体。很多代码清单和用粗体显示的重要概念都有代码注释。有时，代码清单中还会给出数字符号，而相应的解释在下文中。

本书的源代码可以从Manning的网站下载：www.manning.com/SassandCompassinAction。代码如有变动会及时更新到：https://github.com/pengwynn/sass-and-compass-in-action。

作者在线

购买本书，你可以访问由Manning出版社运行的一个内部论坛，可以在这里发表关于本书的评论，询问技术问题，并得到作者和其他用户的帮助。为了访问论坛并注册，请打开www.manning.com/SassandCompassinAction。网页上介绍了注册后进入论坛的方法，可以获得什么帮助，以及论坛的规则是什么。

Manning出版社承诺在读者之间以及读者和作者间建立有意义的交流平台。但并不承诺本书的部分作者一定会参与其中，作者在论坛上所做的一切贡献都是自愿的（且是无偿的）。我们建议你尝试提出一些富有挑战的问题，这样可以激发他们的兴趣。

只要书一出版，Author Online论坛和以前的讨论内容就可以从出版社的网站访问。

关于封面插图

本书封面插画的标题为“斯兰卡”，她是居住在盖尔河谷的斯拉夫人。斯洛文尼亚人把这条河称为Zilja。这条河起源于奥地利南部，流经尤里安阿尔卑斯山很多风景如画的地方。该插画来自克罗地亚斯普利特民族博物馆2008年出版的Balthasar Hacquet的《图说西南及东汪达尔人、伊利里亚人和斯拉夫人》（*Images and Descriptions of Southwestern and Eastern Wenda, Illyrians, and Slavs*）的最新重印版本。

Hacquet（1739—1815）是一名奥地利内科医生及科学家，他花费数年时间去研究各地的植物、地质和人种，这些地方包括奥匈帝国的多个地区，以及伊利里亚部落过去居住的（罗马帝国的）威尼托地区、尤里安阿尔卑斯山脉及西巴尔干等地区。Hacquet发表的很多论文和书籍中都有手绘插图。

Hacquet出版物中丰富多样的插图生动地描绘了200年前西阿尔卑斯和巴尔干西北地区的独特性和个体性。那时候相距几英里的两个村庄村民的衣着都迥然不同，当有社交活动或交易时，不同地区的人们很容易通过着装来辨别。从那之后着装的要求发生了改变，不同地区的多样性也逐渐消亡。现在很难说出不同大陆的居民有多大区别，比如，现在很难区分斯洛文尼亚的阿尔卑斯山地区或巴尔干沿海那些美丽小镇或村庄里的居民与欧洲其他地区或美国的居民。

Manning出版社利用两个世纪之前的服装来设计书籍封面，以此来赞颂计算机产业所具有的创造性、主动性和趣味性。正如本书封面的图片一样，这些图片也把我们带回到了过去的生活中去。

致　　谢

如果没有Hampton Catlin，我们不可能写这本关于Sass和Compass的书。对于我们许多人来说，是Sass让CSS重焕青春。这个过程中，尽管语法在变化，但是扩展CSS以使它变得更加易用、高效的精神从未改变。Hampton Catlin独到的眼光和努力在Sass的项目和社区留下了不可磨灭的印记。

Chris Eppstein，也是本书的作者之一，请接受我们最真诚的感谢。如果不是你在过去几年不辞劳苦地扩展和维护Sass与Compass，社区绝对不会如今天这么壮大。

我们还要感谢Manning的出版团队，为了让这本书出版，他们与我们一起奋战，共同走过了一段非常漫长的旅程。通常，为一个开源项目写书是非常难的，因为项目长期处于发展之中。最终能把这本书交到那些希望它们的前端工具达到更高水平的开发和设计人员手中，我们激动不已。

最后，要特别感谢技术校对Matt Martini。Matt Martini在本书即将付印之前还仔细检查了最后一次校样。同时，特别感谢本书的审稿人，他们在本书写作的不同阶段，反复阅读了各章内容并提出了宝贵的反馈意见，他们是：Adam Michela、Adam Yonk、Andrea Ferretti、David A. Mosher、David Landau、Ezekiel Templin、Graham Ashton、Jacob Rohde、Jake Stutzman、James Hafner、Jason J. W. Williams、Jeremiah Stover、Jeroen van Dijk、Ken Paulsen、Kerrick Long、Kevin Sylvestre、Kyle Wild、Ron Chloupek、Ryan Kelln和William Dodson。

Wynn netherland个人致谢

另外，我还要感谢我的太太Polly，她同时要面对我的另一本书的写作和按时交稿的压力。感谢她爱上我这样疯狂的人。

感谢Jason J. W. Williams，他也是Manning的技术书作者，他为我提供了大量的专业支持，还分享他涉及多种语言的创作工具。

目　　录

第三部分 来到生产环境

Part 1 第一部分

认识 Sass 和 Compass

本书的第一部分会带你初步认识Sass和Compass，了解Sass的核心特性并学习编写动态样式的一些原则。第1章首先介绍编写动态样式的概念，以及让动态样式开发更高效的相关开发原则。你将看到Sass如何通过选择器嵌套和变量来帮助避免重复，以及通过使用继承和混合器让你能够更加高效地重用通用样式。我们将学习Compass框架，看看它怎样通过提供模式和工具让站点的样式编写更加顺畅和高效。

第2章将带你认识Sass语法以及Sass的许多强大功能。我们共同探讨如何在Sass中使用变量，并学习变量的作用域。本章将介绍如何通过嵌套选择器和子属性让样式表更加整洁和易读。你还会了解Sass怎样通过扩展原生CSS的`@import`实现将许多样式合并到一个文件的功能，这样你就可以把你的样式分散到多个更小的、可管理的文件中。我们一起看看如何通过使用混合器在避免重复的同时共享通用的样式，以及如何通过给混合器传参，在混合器中使用变量，在保留样式类型的同时可以轻松定制样式。你将学习使用另一个可以减少重复的方法：选择器继承`@extend`，以及何时使用继承，何时使用混合器，使用它们的最佳实践。

读完前两章，你对Sass语法就不再陌生，并且对于如何改进样式有了灵感。而且，你将对动态样式表的真谛了然于胸。下一部分将从理论转到实战，使用Sass和Compass来解决一些真实世界中的问题。

第 1 章

Sass和Compass让样式表重焕青春

本章内容
- ❑ 开始学习Sass和动态样式表
- ❑ 用Sass更高效地写样式表
- ❑ Compass简介
- ❑ 用Compass迎接工程实践中的样式挑战

Sass是CSS3语言的扩展，它能帮你更省事地写出更好的样式表，使你摆脱重复劳动，使工作更有创造性。因为你能更快地拥抱变化，你也将敢于在设计上创新。你写出的样式表能够自如地应对修改颜色或修改HTML标签，并编译出标准的CSS代码用于各种生产环境。Sass引擎是用Ruby写的，对此你无需在意，除非要修改这个引擎的Ruby代码。

这本书适合两类读者，并尝试挖掘出他们之间的共同点。如果你发现自己两类都符合，那再好不过了。

对于Web设计师小伙伴们：你熟记着Adobe应用的各种快捷键，仅根据RGB值就能选出互补色。你或许（或许没有）戴着一副黑框眼镜，很可能用一杯咖啡或茶和最新的*Smashing Magazine*开始新的一天。你十分了解jQuery很危险，却不明白为什么开发小伙伴会笑你把CSS当做一门语言。

我们将帮你摆脱这些繁冗乏味的琐事，让你做最擅长的事——创新。我们知道你对样式重置、排版比例、颜色调节以及布局很有想法。我们将向你展示怎样通过更少的重复劳动，更快地写出样式表。从此，你将在图形软件上费时更少，而在样式表上做得更多。

对于前端开发同学：你们能够在Photoshop设计稿上切图，然后转化为语义化的HTML和CSS。但是有个问题，你的服务端模板那么符合DRY原则，即“别做重复劳动”（Don't Repeat Yourself），但你的样式表也太不符合这个原则了吧！而且，随着项目规模的增长，你发现组织样式表成了一大挑战。要是能像写项目中其他代码那样，使用变量、可复用模块以及控制语句来写样式表就好了。别担心，Sass正好满足了你的需求。

本章将介绍Sass的强大特性，比如选择器嵌套、变量、混合器、选择器继承，以及Compass

怎样利用这些特性构建可复用的设计模式，从而使你摆脱重复性的体力劳动，专注于设计样式而不是实现样式。如果你还没有安装Sass，请参照附录A中给出的步骤安装Sass。如果你正在咖啡馆用iPad阅读本书，那仍然可以在线运行那些基本示例：http://Sass-lang.com/try.html。

1.1　开始学习 Sass

在开始研究示例之前，有必要明确几点。Sass既不是银弹也不是精灵粉。它不能帮你立刻选出颜色，做好排版和布局，但能帮你更快地实现你的想法并减少麻烦。在开始学习语法和特性之前，先看一下工作流的概况图。使用Sass时，Sass引擎将你的样式表源文件编译成100%纯CSS，如图1-1所示。

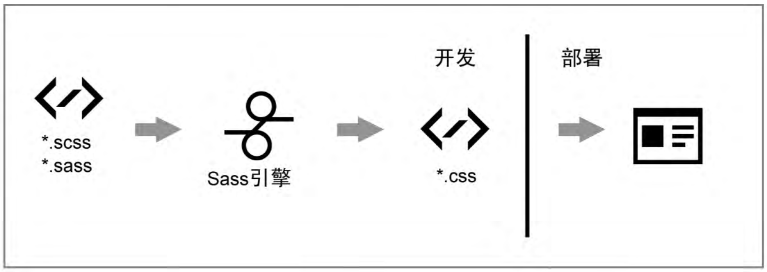


图1-1　Sass编写和编译的工作流

从命令行到服务端集成框架，再到图形用户界面（GUI）工具，虽然有各种选择来运行Sass引擎，但这一工作流中的关键环节是输出CSS。然后你可以像平常那样部署静态的CSS文件，而Sass的语言特性使你能更快地写出具有高可维护性的CSS代码。

1.1.1　从CSS到Sass

如果你精通CSS，很快就能上手Sass。Sass更专注于怎样创造优雅的样式表，而非其内容。我们会介绍一些为你提供CSS最佳实践的工具（如Compass），不过归根结底，如果你拥有扎实的CSS基础即能从本书中获益匪浅。在计算机领域，错进，错出（garbage in, garbage out）原则总是成立的。如果你需要一本CSS入门书，请参阅Manning出版的*Hello! HTML5 and CSS3*。

Sass支持两种语法。最初的缩进式语法使用.sass作为文件扩展名，且这种语法对空格敏感，所以选择器下面的属性要有缩进，而不用花括号括起来。每个属性之间通过换行来分隔，而不是分号：

```
h1
  color: #000
  background: #fff
```

SCSS即Sassy CSS在Sass 3.0中被引入，它是CSS3的超集。SCSS文件使用.scss作为文件扩展名，且到处都是熟悉的花括号和分号。

```
h1 {color: #000; background: #fff}
```

以上展示了这两种语法的主要不同，其他的不同点在附录C中讨论。

这两种语法Sass都会继续支持。你甚至可以在同一个项目中混合使用，只要不在同一个文件中使用即可。选择一种对你和你的团队而言适合的语法是很重要的。如果你周围的人都用Python或Ruby，那么可能空格敏感的缩进式语法会很适合。如果你的团队与外面的设计机构做生意，那么使用Sassy CSS将更容易成功。

除了学习CSS技巧和掌握Sass语法，以动态的思维去看待样式表也很重要。

1.1.2　动态思维

现在除了简单的说明书式的网站，谁还会去写很多静态的HTML？你把HTML文件以模板的形式提供给博客引擎、内容管理系统（CMS）或应用程序的框架去做预处理，然后动态地“填入”标签和内容。这些工具为HTML注入了生命，我们无法想象没有它们互联网将会变成怎样。所以*你为什么还要再写静态的样式表呢？*你将会看到那些被用来动态创建静态标签的特性，也可以用来动态创建静态样式表。何为动态创建静态样式表？意思即：当你写样式表时，不再受限于浏览器是怎样理解样式表的。通过条件判断、可复用的代码片段、变量以及其他的各种工具，你能够让样式表活起来。改变一个网站的布局和颜色风格变得如此简单，只要稍微调整一些变量就可以了。当然，虽然Sass允许你以一种时髦的方式来动态地写样式表，但产出的依旧是100%静态的CSS。一旦你开始使用动态样式表来工作，就能遵循内心深处一直呐喊的声音“*别做重复劳动*”。

1.1.3　别做重复劳动

Sass为样式表的作者提供了强大的工具，帮他们摆脱在写CSS时那些一遍又一遍的乏味劳动。Sass的很多特性都信奉广为流传的编程格言“*别做重复劳动*”，让你既能在写样式表时减少重复劳动，又能减少编译出的样式表中的冗余。创建样式表时，重复就是一种危险信号。你应该时常问自己：*我怎样更高效地工作，而不仅仅是更卖力？*接下来几节将向你展示怎样使用Sass来提升样式表的可复用性。

1.2　Sass 你好：消除样式表冗余

写到这里我们已经唠叨很多遍“别做重复劳动”了。那么充满冗余的样式表是什么样的呢？请看代码清单1-1中的CSS。

代码清单1-1　一个需要消除冗余的糟糕样式表

```
h1#brand {color: #1875e7}

#sidebar { background-color: #1875e7}

ul.nav {float: right}
```

```
ul.nav li {float: left;}
ul.nav li a {color: #111}
ul.nav li.current {font-weight: bold;}

#header ul.nav {float:right;}
#header ul.nav li {float:left;margin-right:10px;}
#footer ul.nav {margin-top:1em;}
#footer ul.nav li {float:left;margin-right:10px;}
```

即使是这样一个极其简单的例子，但显而易见还是有很多重复劳动。如果市场团队想要把那可爱的蓝色阴影从`#1875e7`调整成`#0f86e3`，将会发生什么？当然，只有两处修改的话还是可控的，但如果有十几处或更多处修改散落在不同的文件当中，那么“查找/替换”这种方式就会显得很原始，不是吗？另外，8个`ul.nav`在只有10行的样式表中也显得太多了。

在接下来的小节中，你将会学到一些写起来犹如夏日微风般惬意的语法糖，它们将会帮你摆脱样式表中的重复劳动，它们包括变量、混合器、选择器嵌套以及选择器继承。如果你觉得学习进度太快了，别怕，第2章会深入介绍每个概念。

1.2.1 通过变量来复用属性值

你是否在样式表中用过“查找/替换”来修改十六进制的颜色值？使用Sass，你可以把属性值赋给变量，然后在一个地方管理散落在各处的属性值，几乎任意属性值都可以赋给变量，比如颜色，边框大小：

```
$company-blue: #1875e7;

h1#brand {
color: $company-blue;
}

#sidebar {
background-color: $company-blue;
}
```

变量名以`$`符号开头，可包含所有可用作CSS类名的字符，包括下划线和中划线。在这个简单的例子中，如果你想要微调蓝色阴影，只需要修改变量赋值的那一处地方，使用变量的其他地方就会自动更新。

如果你有开发背景，会对变量感觉自然而然。如果你只有设计背景，第一眼看到变量可能会觉得有点陌生。但它真的不是什么新鲜事物。你已经在CSS中使用过很多有名字的属性值，比如`blue`、`green`、`inherit`、`block`、`inline-block`、`serif`以及`sans-serif`。把变量想象成这些特殊的属性值。接下来，我们将展示怎样使用选择器嵌套来创建多层级的选择器，从而减少你的敲写工作。

1.2.2 使用嵌套来快速写出多层级的选择器

先给你讲个故事。有个得克萨斯人，他的工作是画出高速公路中间的虚线。第一个星期他是

做得最好的那个，画了10英里。但接下来工作成果迅速降低，第二个星期他画了五英里，第三个星期只画了两英里，第四个星期只画了可怜的一英里。主管问他遇到了什么困难。他回答道："哎，回到桶边的距离总是变得越来越远。"

这感觉正像使用多层级的选择器。请看下面的CSS：

```
ul.nav  {float: right}
ul.nav li {float: left;}
ul.nav li a {color: #111}
ul.nav li.current {font-weight: bold;}
```

Sass能帮你减少一些重复劳动。在第1章的代码示例文件夹中找到文件。1.1.2.nesting.scss，或者将代码清单1-2保存为一个文本文件，创建你自己的代码文件。

代码清单1-2　嵌套CSS的选择器

```
ul.nav {
  float: right;

  li {
    float: left;
    a {
      color: #111;
    }
    &.current {
      font-weight: bold;
    }
  }
}
```

在终端中运行如下`sass`命令，并传入文件的路径：

```
Sass 1.2.nesting.scss
```

你将会在终端输出中得到如代码清单1-3所示的CSS结果。

代码清单1-3　通过使用选择器嵌套生成的CSS

```
ul.nav {
  float: right; }
  ul.nav li {
    float: left; }
    ul.nav li a {
      color: #111; }
    ul.nav li.current {
      font-weight: bold; }
```

除了格式上的一些不同，它跟我们一开始的CSS是相同的。（别在意目前的格式，后面会讨论更多输出选项。）

使用Sass，你可以将规则嵌套起来，并避免在选择器中重复写相同的元素。这不仅节约了时间，还有一个好处是，如果你稍后要把`ul.nav`从一个无序列表改成有序列表，你只需要修改一处。这个例子中的最后一个选择器原理相同。&符号是父选择器。在这个例子中，`&.current`相

当于li.current。如果将来li改成了其他的元素标签，在这个元素内的current类依然命中这里的样式。现在你已经见过了怎样使用变量来复用属性值，怎样使用嵌套来写多层级的选择器，接下来让我们一鼓作气，看看混合器。

1.2.3　使用混合器来复用一段样式

变量使你能够复用属性值，但如果你想要复用一大段规则呢？传统的做法是，如果在样式表中发现重复，就会把公共的规则抽离出来放到新的CSS类中。

代码清单1-4　传统的CSS重构

```
ul.horizontal-list li {
  float: left;
  margin-right: 10px;
}

#header ul.nav {
  float: right;
}

#footer ul.nav {
  margin-top: 1em;
}
```

接下来你需要给元素ul.nav添加额外的类名horizontal-list。这种方式能够有效地工作，但如果你想要使类保持语意的同时依旧可复用，怎么实现呢？

打开1.1.2.mixins.scss，或者自己创建一个新文件，代码清单1-5是第二个例子。

代码清单1-5　通过使用@mixin和@include重用代码

```
@mixin horizontal-list {
  li {
    float: left;
    margin-right: 10px;
  }
}

#header ul.nav {
  @include horizontal-list;
  float: right;
}

#footer ul.nav {
  @include horizontal-list;
  margin-top: 1em;
}
```

顾名思义，Sass混合器用于将一些规则混入到别的规则当中。使用@mixin命令，你可以将水平列表的规则抽取出来放到一个合适的命名混合器中。然后使用@include命令，将这些规则引入进来。因此你不再需要.horizontal-list类，因为那些规则已经被混入到ul.nav的规则

当中，输出的CSS如代码清单1-6所示。

代码清单1-6　混合器帮助你消除冗余代码

```
#header ul.nav {
  float: right;
}

#header ul.nav li {
  float: left;
  margin-right: 10px;
}

#footer ul.nav {
  margin-top: 1em;
}

#footer ul.nav li {
  float: left;
  margin-right: 10px;
}
```

更便利的是混合器和变量的结合，这才是混合器的强大之处；也就是说，能够根据参数来决定使用的样式，从而使混合器更具可复用性。举个例子，假如你想要修改水平列表每个条目之间的间距，怎么使用混合器快速实现呢？找到代码示例1.1.2.2.mixins-parameters.scss，按照代码清单1-7对它进行修改。

代码清单1-7　在混合器中应用变量

```
@mixin horizontal-list($spacing: 10px) {
  li {
    float: left;
    margin-right: $spacing;
  }
}

#header ul.nav {
  @include horizontal-list;
  float: right;
}

#footer ul.nav {
  @include horizontal-list(20px);
  margin-top: 1em;
}
```

OK，我们给混合器添加了一个参数`$spacing`，其默认值为`10px`。参数这一概念跟我们之前看到的变量没有什么不同。在这个例子中通过使用默认值，头部的水平导航列表就得到默认的间距。在底部的导航列表中，通过传入值`20px`，从而增加列表元素之间的间距，输出的CSS如代码清单1-8所示。

代码清单1-8　通过使用混合器生成的最终CSS

```
#header ul.nav {
  float: right;
}

#header ul.nav li {
  float: left;
  margin-right: 10px;
}

#footer ul.nav {
  margin-top: 1em;
}

#footer ul.nav li {
  float: left;
  margin-right: 20px;
}
```

混合器能够让你复用一段属性，从而为你节省大量时间。不过聪明的读者或许发现了输出的样式表中有冗余，这是因为混合器在每一个被包含进来的地方，都会就地复制一段样式。别担心，Sass总留有一手。选择器继承就可以解决这个问题。

1.2.4　使用选择器继承来避免重复属性

如你所见，混合器能够在手写的样式表中有效地避免重复。但是，因为规则都混入到其他类中，所以在输出的样式表中不能完全避免重复。因为输出的CSS文件大小很重要，于是Sass引入了另一种稍微有点复杂的方式，让输出的CSS完全避免重复。选择器继承的意思就是让一个选择器能够复用另一个选择器的所有样式，但又不重复输出这些样式属性。看个例子你就明白了，比如，为表单的错误信息设置一些样式。

代码清单1-9　修饰错误信息的CSS

```
.error {
  border: 1px #f00;
  background: #fdd;
}

.error.intrusion {
  font-size: 1.2em;
  font-weight: bold;
}

.badError {
  @extend .error;
  border-width: 3px;
}
```

通过选择器继承，可以让`.barError`继承父类`.error`，也就是复用父类的所有样式，编译输出的结果如代码清单1-10所示。

代码清单1-10　通过选择器继承减少冗余

```
.error, .badError {
  border: 1px #f00;
  background: #fdd;
}

.error.intrusion,
.badError.intrusion {
  font-size: 1.2em;
  font-weight: bold;
}

.badError {
  border-width: 3px;
}
```

在这个例子中，同时定义了`error`和`badError`类是有意义的，因为两者都需要在HTML中使用，但有时父类并不需要在HTML中使用。于是在Sass 3.2中引入了占位选择器，它支持在使用选择器继承的同时，不编译输出HTML中并不使用的父类，如代码清单1-11所示。

代码清单1-11　在选择器继承中使用占位选择器

```
%button-reset {
  margin: 0;
  padding: .5em 1.2em;
  text-decoration: none;
  cursor: pointer;
}

.save {
  @extend %button-reset;
  color: white;
  background: #blue;
}

.delete {
  @extend %button-reset;
  color: white;
  background: red;
}
```

占位顾名思义，继承了`%button-reset`的选择器在输出的CSS中占据了`%button-reset`的位置，如代码清单1-12所示。

代码清单1-12　通过选择器继承最终生成的CSS

```
.save, .delete {
  margin: 0;
  padding: .5em 1.2em;
  text-decoration: none;
  cursor: pointer;
}
```

```
.save {
  color: white;
  background: #blue;
}

.delete {
  color: white;
  background: red;
}
```

占位选择器能把常用的样式保存到一处，且不影响任何一个类名，使你能够放心使用。当然如果一个占位选择器没有被继承，其中的样式就不会被编译到CSS当中，以减少生产环境中样式表的无用样式，使其更小。

只需要一点小小的规划，选择器继承就能够让你摆脱写Sass代码时的重复劳动，并让输出的CSS文件保持整洁。现在你已经看到Sass怎样帮你避免重复劳动了，下一节将介绍Compass的功能。

1.3　Compass 是什么

简单来说，Compass是一个强大的Sass框架，它的设计目标是顺畅、高效地装扮互联网，使用它的人可写出可维护性更高的样式表，而它已被授权给由设计师和开发者组成的社区来共同构建和分享。互联网应用的开发框架有很多，比如Rails就是面向Ruby的一个框架，类似地，Compass就是面向Sass的一个框架，它有一套实用的工具，并在长期的实战中总结出了最佳实践。

Compass由三个主要部分组成：混合器和实用工具的类库，能够集成到应用开发环境中的开发系统，以及一个用于构建框架和扩展的平台。让我们将本章之前提到的工作流概况图扩展一下，如图1-2所示，我们可以看到Compass如何融入到你的开发工作流中。

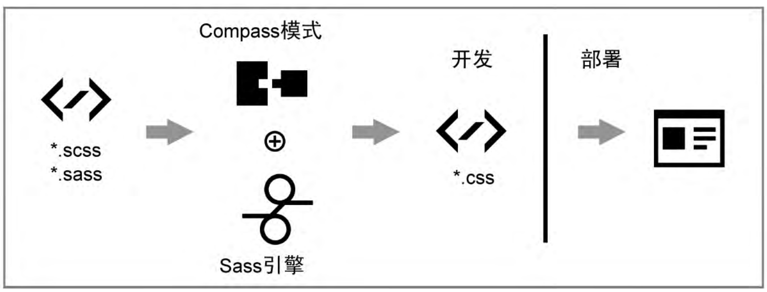


图1-2　Compass工作流

1.3.1　Compass库

Compass自身包含了很多由Sass混合器和函数构成的模块，所有的这些模块在Compass的官网上都有详细的说明和示例。这些库会帮助你解决跨浏览器兼容问题，以及提供给你很多已经被证明过的优秀设计模式，比如说重置、网格布局、列表样式、表格辅助器和垂直韵律等。Compass

同时提供了CSS3的帮助函数，帮你填充厂商前缀、抽离不同浏览器在实现同一CSS3属性时的不同，并让你写起CSS3属性来更得心应手。

Compass能非常灵巧地完成一些任务，比如说直接从文件系统读取你的图片然后输出对应的样式到你的CSS中。资源URL辅助函数让你能够轻松地切换到使用CDN网络进行项目部署，而无需重写你的样式引用。Compass甚至能自动帮你把同一目录的图片合成精灵图，帮你计算好单个图片的引用位置并自动输出到CSS文件中。

以上的任务你都可以通过自己的方式完成，但是经由设计社区验证的Compass方案能让你在更短的时间内完成更多的任务。

Compass的核心样式框架并不能让你的网站更美。实际上，所有的核心框架的特性都是独立于设计的，以便你能在任何风格的网站设计中使用。像所有的流行事物一样，网站的设计美学也在不断地过时和更新。所以提供精心设计的网站这个任务就留给了Compass社区的前端开发人员和设计人员，通过使用Compass的插件来实现。

1.3.2　简单的Compass样式项目

Sass和Compass都是用Ruby写的且起源于Ruby on Rails社区，但是Compass提供了工具和配置项，使在不基于Ruby的项目中编写Sass样式表也非常简单。无论你只是简单地写个html文件，还是把Sass集成到大型框架（比如Django、Dral或者.NET），Compass都能胜任。

Compass明白你不单单是在写样式，你是在构建设计。正因为如此，Compass需要知道你在哪里存放图片、字体和JavaScript文件，以便轻松管理并在样式中引用这些文件。举例来说，Compass会自动帮你构建精灵图，并在CSS中正确引用它们；假如你通过`image-url()`辅助器引用了一个并不存在的图片，Compass会告警；Compass可以在你的CSS中内嵌必要的图片和字体，免去浏览器的一些工作。



图1-3　一个标准的Compass项目

1.3.3　社区生态系统

如果你有多年的Web开发经验，一定忘不了JavaScript框架诞生之前的黑暗日子。那真是一个可怕的时期——DOM上一个最小的诡异问题都要耗费你几个小时去排查。而现如今，JavaScript框架帮你解决了浏览器之间不一致的问题，并形成了一套基础的封装，让你的代码可以轻松移植到别人的项目上。多亏了Web开发社区辛苦努力的付出，现在用JavaScript开发简直就是一种享受。

作为Sass的一个框架，Compass致力于为开发者和设计人员提供一个基础架构，从而分享他们的库和框架，为大家参与开源样式开发推力助航。以前，分享一段CSS代码必须在博客文章中内嵌一段代码和示例文件，这样的日子正快速逝去，那种策略还有一个很大的弊端就是：随着时间的流逝，这段代码的作者很难再去解决bug或者增加新的功能。而有了Compass，样式库可以像其他软件一样分发，这就意味着修复bug或者添加对最新浏览器的支持无非就是升级和重新编译你的样式表。

很多社区成员会把他们的很多代码提炼打包到Compass中封装，然后供其他人直接使用，避免其他人在写静态CSS代码时还得重新编写复杂嵌套的代码（第10章会介绍如何编写你自己的Compass组件）。响应式布局、段落比例、自定义动画、精美的按钮、图片元素集，以及色板都可以通过Sass来编写，然后扩展到Compass中。利用Compass扩展，就无需构建无聊的基础代码，从而把更多的精力用于实现网站的独特功能。在你从Sass初学者向专家过渡的过程中，如果你一如既往地认可和感谢Sass、Compass和社区所做的工作，那么作为回报，你可以把你的辛苦工作成果分享给其他人。

1.4　创建一个 Compass 项目

如果你还没有安装Compass，可以参考附录A的指引进行安装。安装好以后，第一件事就是创建一个Compass项目。

同其他优秀的命令行接口工具（CLI）一样，Compass提供了非常丰富的帮助信息来帮我们了解Compass的各项功能。测试一下Compass是否安装成功。打开命令行进入一个新的样式项目目录，运行`compass help`。如果成功打印出了帮助信息和命令行参数信息，说明Compass安装好了。如果没有的话，请重新按照附录A的指引进行安装，然后重新尝试。

下面从创建一个新的Compass项目开始，新项目包含了一个配置文件和Sass源文件、CSS输出文件等。将该文件命名为*sample*。

```
compass create sample
```

新目录下会生成这么几个文件：

```
total 8
drwxr-xr-x  6 wynn  staff  204 Jan  3 12:11 .
drwxr-xr-x  3 wynn  staff  102 Jan  3 12:12 ..
drwxr-xr-x  4 wynn  staff  136 Jan  3 12:11 .Sass-cache
-rw-r--r--  1 wynn  staff  315 Jan  3 12:11 config.rb
drwxr-xr-x  5 wynn  staff  170 Jan  3 12:11 Sass
drwxr-xr-x  5 wynn  staff  170 Jan  3 12:11 stylesheets
```

使用默认配置，Compass已经帮我们默认生成了一个`config.rb`配置文件，一个名为`sass`的目录，里边存放Sass的源文件，一个名为`stylesheets`的目录，里边存放Sass编译后生成的CSS文件。如果你想了解`config.rb`里边的所有配置项，请参考附录B。现在我们将使用默认配置来演示如何用Compass解决实际工作中遇到的CSS问题。

1.5　使用 Compass 解决真实的 CSS 问题

你已经知道了如何使用Compass创建一个基本的项目，接下来我们看看如何使用Compass应对我们几乎每天都面对的样式挑战。接下来的几小节将通过使用Compass的内置模块（Sass的函数和其他的特性组合而成）来实现CSS重置、网格布局、表格格式化和CSS3的一些特性。

1.5.1　通过重置来保持样式表现一致

通过Eric Meyer和其他的一些倡议者的努力，在开始创建样式之前添加一个重置文件已经变得非常普及。如果你曾经使用过CSS网格布局框架，里边就包含了CSS重置，只是你不知道而已。CSS重置无非就是把浏览器对于各种元素添加的CSS属性去掉，给开发者提供一个完全无干扰的开始来添加你想要的样式。

Eric的经典重置样式如代码清单1-13所示。

代码清单1-13　经典CSS重置

```
/* v1.0 | 20080212 */

html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, font, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td {
  margin: 0;
  padding: 0;
  border: 0;
  outline: 0;
  font-size: 100%;
  vertical-align: baseline;
 background: transparent;
}
body {
  line-height: 1;
}
ol, ul {
  list-style: none;
}
```

```
blockquote, q {
  quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
  content: '';
  content: none;
}

/* 不要忘了定义focus样式! */
:focus {
  outline: 0;
}

/* 记得通过其他修饰方式来突出新插入内容! */
ins {
  text-decoration: none;
}
del {
  text-decoration: line-through;
}

/* 同时需要在表格元素标签上添加 'cellspacing="0"' */
table {
  border-collapse: collapse;
  border-spacing: 0;
}
```

你或许已经注意到Compass项目默认生成的screen.css文件了，这个文件就是基于Erics的reset脚本衍生出来的。只要通过以下代码添加它，就可以让你的样式在各个浏览器里边起点一致：

```
@import "compass/reset"
```

就这一条命令，其实干了很多事情，我们一条一条来说。你通过使用Sass的引入命令`@import`引入Compass的重置模块。一个模块就是Compass框架中独立的一部分，可被随意引用添加到你的项目中。通过加入这行代码，你生成的CSS文件中就会包含CSS重置部分代码，如代码清单1-14所示。

代码清单1-14 通过引入CSS重置生成的文件

```
html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, font, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td {
  margin: 0;
  padding: 0;
  border: 0;
  outline: 0;
  font-weight: inherit;
  font-style: inherit;
```

```
  font-size: 100%;
  font-family: inherit;
  vertical-align: baseline;
}

body {
  line-height: 1;
  color: black;
  background: white;
}

ol, ul {
  list-style: none;
}

table {
  border-collapse: separate;
  border-spacing: 0;
  vertical-align: middle;
}

caption, th, td {
  text-align: left;
  font-weight: normal;
  vertical-align: middle;
}

q, blockquote {
  quotes: "" "";
}
q:before, q:after, blockquote:before, blockquote:after {
  content: "";
}

a img {
   border: none;
}
```

大多数Compass模块并不会无故在你的样式表中添加样式，不要让CSS重置模块误导了你，Compass重置模块位于样式表的前边，只要被引入就会执行`global-reset`这一混合器。这个混合器的写法如代码清单1-15所示。

代码清单1-15　CSS `global-reset`混合器

```
@mixin global-reset {
  html, body, div, span, applet, object, iframe,
  h1, h2, h3, h4, h5, h6, p, blockquote, pre,
  a, abbr, acronym, address, big, cite, code,
  del, dfn, em, font, img, ins, kbd, q, s, samp,
  small, strike, strong, sub, sup, tt, var,
  dl, dt, dd, ol, ul, li,
  fieldset, form, label, legend,
  table, caption, tbody, tfoot, thead, tr, th, td {
```

```
    @include reset-box-model;
    @include reset-font; }
  body {
    @include reset-body; }
  ol, ul {
    @include reset-list-style; }
  table {
    @include reset-table; }
  caption, th, td {
    @include reset-table-cell; }
  q, blockquote {
    @include reset-quotation; }
  a img {
    @include reset-image-anchor-border; } }
```

这里我们注意到Compass使用了之前提到过的Sass的`@mixin`混合器和`@include`包含特性来构建reset模块。在`global-reset`之外，reset模块还提供了一系列有特定用途的重置混合器，其中有一个是专为HTML5元素服务的。通过在你的Sass文件中添加`@include reset-html5`命令，输出文件中会生成额外的CSS规则来对HTML5的元素进行基本的样式修改，如代码清单1-16所示。

代码清单1-16　使用了HTML5重置模块的输出代码

```
article, aside, canvas, details, figcaption, figure,
footer, header, hgroup, menu, nav, section, summary {
  margin: 0;
  padding: 0;
  border: 0;
  outline: 0;
  display: block;
}
```

如果你想了解更多的Compass模块，请访问Compass官网查看在线文档。现在你已经熟练掌握了重置模块，接下来我们看看如何使用Compass的网格布局框架来更加高效地编写CSS。

1.5.2　不用计算创建布局

这几年CSS的主要趋势就是越来越流行网格框架，比如说Blueprint和960网格系统（如图1-4所示）。网格布局奠定了好的平面设计的基础，在这些年发展的愈发成熟。你只需在布局中选择特定数量的列，每列之间的内容自然会被统一地间隙隔开。



图1-4　960px宽的网格系统布局

一般来讲，有了网格布局框架，在写一个优良的列式布局时就无需进行计算。框架通过CSS规则帮你设置好容器元素的宽以及在容器内的每个子列的宽。让我们看一下Blueprint框架里边的脚本是如何写的，如代码清单1-17所示。

代码清单1-17　Blueprint网格布局

```
.container {
  width: 950px;
  margin: 0 auto;
}

/* 建立基本的格式浮动和外间距. */
.column, .span-1, .span-2, .span-3, .span-4, .span-5,
.span-6, .span-7, .span-8, .span-9, .span-10, .span-11,
.span-12, .span-13, .span-14, .span-15, .span-16,
.span-17, .span-18, .span-19, .span-20, .span-21,
.span-22, .span-23, .span-24 {
```

```
    float: left;
    margin-right: 10px;
  }

  /* 一行中的最后一列需要添加这个类修饰. */
  .last { margin-right: 0; }

  /* 通过使用下边这些类来设定列宽. */
  .span-1 {width: 30px;}

  .span-2 {width: 70px;}
  .span-3 {width: 110px;}
  .span-4 {width: 150px;}
  .span-5 {width: 190px;}
  .span-6 {width: 230px;}
  .span-7 {width: 270px;}
  .span-8 {width: 310px;}
  .span-9 {width: 350px;}
  .span-10 {width: 390px;}
  .span-11 {width: 430px;}
  .span-12 {width: 470px;}
  .span-13 {width: 510px;}
  .span-14 {width: 550px;}
  .span-15 {width: 590px;}
  .span-16 {width: 630px;}
  .span-17 {width: 670px;}
  .span-18 {width: 710px;}
  .span-19 {width: 750px;}
  .span-20 {width: 790px;}
  .span-21 {width: 830px;}
  .span-22 {width: 870px;}
  .span-23 {width: 910px;}
  .span-24 {width:950px; margin-right:0;}
```

使用以上这些CSS规则，通过在容器元素上添加container类，并在你希望放置在容器内的子元素上添加span-xx类，就可以创建出一个16列的布局。通过这样的方式进行布局，你能够快速地构建出原型，且不用记住在30和950之间的40的倍数。

那么Compass做了哪些事情来优化提升CSS网格布局呢？首先Compass以混合器的形式提供了对网格框架样式的支持，这样做的好处是按需使用，避免你的HTML标签需要添加额外的类名。其次，也是最重要的，Compass对网格布局的支持方式即是对其他框架的方式改进，这将在第4章详细介绍。

我们通过Blueprint创建一个Compass工程，在命令行中运行下面的命令：

```
compass create my_grid --using blueprint
```

就像在1.4节里边那样，你会在my_grid目录下找到一个全新的Compass工程，只是这次screen.scss文件的内容更多。这个文件有着详细的注释，且在一系列基本布局的样式里边，你可以看到有一个Blueprint模块快速的总览。首先映入眼帘的恐怕是列式布局相关的样式被归类成了混合类，所以不是在HTML标签上添加一个span-8的类名，而是使用Sass的column混合器。

```
@include column($sidebar-columns);
```

还要注意`$sidebar-columns`变量。多亏了Sass，你现在可以通过变量来控制你的布局，多么强大啊。仅仅修改几个Sass文件顶部的变量，就可以快速地构建不同列、不同列间隔和不同边栏大小的原型。如果想在传统的CSS网格框架中做这件事情，你只能自己去计算这些东西，然后在你的HTML标签上修改类名。

这里我们就不展开Blueprint网格布局了，留待第6章直接使用。接下来，通过构建真实的Compass项目来进一步了解Compass的表格辅助器。

1.5.3　通过表格辅助器为表格添加更专业的斑马条纹样式

继续我们对Compass特性的探索，来看看它的表格辅助器。表格辅助器由一系列Sass混合器构成，它使表格样式编写更轻松容易，如代码清单1-18所示。

代码清单1-18　Compass表格辅助器

```
@import "compass/reset"
@import "compass/utilities/tables";

table {
  $table-color: #666;
  @include table-scaffolding;
  @include inner-table-borders(1px, darken($table-color, 40%));
  @include outer-table-borders(2px);
  @include alternating-rows-and-columns($table-color,
             adjust-hue($table-color, -120deg), #222222); }
```

我们一行行来看。首先通过`@import`引入表格辅助器，这样就有了4个可以使用的混合器。`table-scaffolding`混合类提供了最基本的样式修饰，用于已经过CSS重置处理的`th`和`td`元素，其中包括我们常用的一些修饰，比如说数字列会有一个右对齐。我们看看这个混合器的源码，如代码清单1-19所示。

代码清单1-19　表格辅助器中的混合器

```
@mixin table-scaffolding {
  th {
    text-align: center;
    font-weight: bold; }
  td,
  th {
    padding: 2px;
    &.numeric {
      text-align: right; } } }
```

`inner-table-borders`和`outer-table-borders`混合器正如其名，用于给表格以及表格内的单元格添加边框。

最后，`alternating-rows-and-columns`混合器提供了一种更简单的方式，来给你的HTML表格添加斑马条纹样式。你或许会问为什么不用`:nth-child`、`:even`或者`:odd`等CSS伪

类选择器呢？这是个好问题。其实，在底层Compass就是通过这种方式实现的。但是这个混合器除了样式上颜色的交错等修饰还会提供一些基于类名的脚本支持。我们看一下它的源码，如代码清单1-20所示。

代码清单1-20　按照行或列更改颜色的混合器

```
@mixin alternating-rows-and-columns(
        $even-row-color,
        $odd-row-color,
        $dark-intersection,
        $header-color: white,
        $footer-color: white) {
   th {
     background-color: $header-color;
     &.even, &:nth-child(2n) {
     background-color: $header-color - $dark-intersection; }
  }
  tr.odd {
    td {
      background-color: $odd-row-color;
      &.even, &:nth-child(2n) {
        background-color: $odd-row-color - $dark-intersection; }
      }
  }
  tr.even {
    td {
       background-color: $even-row-color;
       &.even, &:nth-child(2n) {
         background-color: $even-row-color - $dark-intersection; }
       }
    }
    tfoot {
      th, td {
        background-color: $footer-color;
        &.even, &:nth-child(2n) {
           background-color: $footer-color - $dark-intersection; }
      }
    }
```

注意，颜色值不仅仅是一个变量，它们之间还会用来进行一些计算，以便提供合适的对比来保证网页的可读性。下一章你将学到更多关于Sass处理变量和计算的知识。我们继续往下看，了解Compass是如何实现在写CSS时无需再写厂商前缀的。

1.5.4　CSS3 属性无需再写厂商前缀

当CSS3的新属性被越来越多的浏览器新版本所采纳时，设计师们兴奋极了，因为他们可以用新的CSS属性实现以前需要非常愚蠢的样式技巧才能实现的任务。令人异常激动的是，现在用几行非常简单的CSS就能实现边框圆角，且不用在意随之而来的厂商前缀。厂商前缀就是浏览器在CSS特性前加的`-webkit`和`-moz`等前缀，用于对试验中的CSS特性进行支持。举个最简单的例

子，要给一个<div>元素设置5px的边框圆角，你必须要这么写CSS：

```
.rounded {
  -webkit-border-radius: 5px;
  -moz-border-radius: 5px;
}
```

一般来说，使用Compass的CSS3模块中的边框圆角混合器可以免去你这部分的重复工作。首先在你的Sass文件中引入CSS3模块，然后引入混合器：

```
@import "compass/css3";
.rounded {
  @include border-radius(5px);
}
```

看一下生成的CSS：

```
.rounded {
  -moz-border-radius: 5px;
  -webkit-border-radius: 5px;
  -o-border-radius: 5px;
  -ms-border-radius: 5px;
  border-radius: 5px;
}
```

这样做，不仅避免了重复输入-webkit和-moz，同时也支持了其他的一般常用的厂商命名空间，令你设计的网页可以在更多浏览器中保持良好设计。尽管这一点点重复看上去并不是那么恐怖，可如果你仅仅是想让第四个角是圆角呢？对此，火狐浏览器的开发者们并未就最好的实现方式与业界其他人士达成一致，而惯用的实现方式如下：

```
.rounded-one {
  -moz-border-radius-topleft: 5px;
  -webkit-border-top-left-radius: 5px;
}
```

这个地方，Compass有更好的实现。你可以通过border-corner-radius混合器来指定单个边框圆角：

```
.rounded-one {
  @include border-corner-radius(top, left, 5px);
}
```

生成的CSS如下，火狐的方式也包含在内：

```
.rounded-one {
  -moz-border-radius-topleft: 5px;
  -webkit-border-top-left-radius: 5px;
  -o-border-top-left-radius: 5px;
  -ms-border-top-left-radius: 5px;
  border-top-left-radius: 5px;
}
```

以上只是Compass中有关CSS3的冰山一角的一角的一角。第9章将更深入地接触所有能节省时间的特性。

1.6 小结

在这第1章，我们学习了CSS预处理的示例，并简单介绍了一下Sass的4个关键特性：变量、选择器嵌套、混合器和选择器继承。同时，我们也了解了一些Compass框架中Sass特性在真实世界程序中的应用，包括CSS重置、格式布局、表格样式修饰以及CSS3边框圆角。

下一章将进一步学习Sass的语法，包括颜色函数和脚本支持。当你对Sass掌握得更好一些时，我们将更深入地学习Compass。

第2章

Sass基础语法

2

本章内容

- 通过变量重用颜色、长度以及其他一些值
- 通过规则嵌套让CSS更加结构化
- 通过多文件组织让样式变得更加可维护
- 通过混合器和继承重用整个样式

Sass的SCSS语法是CSS3语法的一个超集。这就意味着如果你会读写CSS，就已经知道了如何读写基本的Sass。

在CSS的基础之上，Sass添加了新特性和新语法，让你能够用更少量的代码更加清楚地表示更多的样式。添加某些Sass特性是为了让那些已掌握CSS但可能从未接触过Sass的人，能轻松地理解样式。当然，大多数人还需给予一些指导，这即是本章的目的。

第一次读Sass文件时，你一开始会只看你熟悉的部分：CSS。所有Sass文件的根本目标都是要像CSS那样设计网站的样式，所以样式定义始终是这个文件中最重要的部分。

只要你理解了样式部分，你就可以看其中用到的Sass特性了。如果有不明白的地方，可以在本书中找到相关解释。看的同时，要常常问自己，*这样做对表示样式有什么帮助？*

写Sass与读Sass很相似，你就当自己是在写CSS。然后你考虑哪些Sass特性用进来可以优化你写的CSS。你是否在很多地方用了同一种颜色（或稍有差异的颜色）？你的选择器是否都以同一ID作为父选择器？类似这样的问题，Sass能帮你优化。

第1章简单地过了一下Sass的主要特性。本章将更加深入、详细地介绍这些特性。我们首先谈一谈变量——Sass中最简单、最基本的重用形式。然后，介绍如何通过选择器嵌套来保持样式的整洁和易读性。接着，介绍如何把大片的样式分拆到不同的文件，然后通过组织文件的形式保持可维护性。与此同时再穿插讲一下未提到的注释，它在我们的代码文件中且对外不可见。最后，我们将学习两种重用大段样式的方法：混合器和继承。前者让一遍遍重用同一类型的样式变得非常简单，而后者用于表达类之间的关系。

首先讨论变量，它是Sass中最基本的重用形式。

2.1　使用变量

Sass让人们受益的一个重要特性就是它为CSS引入了变量。你可以把反复使用的CSS属性值定义成变量，然后通过变量名来引用它们，而无需重复书写这一属性值。或者，对于仅使用过一次的属性值，你可以赋予其一个易懂的变量名，让人一眼就知道这个属性值的用途。

Sass使用$符号来标识变量，比如$highlight-color和$sidebar-width。为什么选择$符号呢？因为它好认、更具美感①，且在CSS中并无他用，不会导致与现存或未来的CSS语法冲突。

变量的最基本方面就是一开始的变量声明，即我们接下来要讨论的内容。

2.1.1　变量声明

Sass变量的声明跟CSS属性的声明很像：

```
$highlight-color: #abcdef;
```

这意味着变量$highlight-color现在的值是#abcdef。任何可以用作CSS属性值的赋值都可以用作Sass的变量值，甚至是以空格分割的多个属性值，如$basic-border: 1px solid black;，或以逗号分割的多个属性值，如$plain-font: "Myriad Pro"、Myriad、"Helvetica Neue"、Helvetica、"Liberation Sans"、Arial和sans-serif; sans-serif;。这时变量还没有生效，除非你引用这个变量——我们很快就会了解如何引用。

与CSS属性不同，变量可以在CSS规则块定义之外存在。当变量定义在CSS规则块内，那么该变量只能在此规则块（或其子级规则块；见2.2节）内使用。如果它们出现在任何形式的{...}块中（如@media或者@font-face块），情况也是如此：

```
$nav-color: #abcdef;

nav {
  $width: 100px;
  width: $width;
  color: $nav-color;
}
```

在这段代码中，$nav-color这个变量定义在了规则块外边，所以在这个样式表中都可以像nav规则块那样引用它。$width这个变量定义在了nav的{ }规则块内，所以它只能在nav规则块内使用。这意味着是你可以在样式表的其他地方定义和使用$width变量，不会对这里造成影响。

只声明变量其实没啥用处，我们最终的目的还是使用它们。上例已介绍了如何使用$nav-color和$width这两个变量，接下来我们将进一步探讨变量的使用方法。

2.1.2　变量引用

凡是CSS属性的标准值（比如说1px或者bold）可存在的地方，变量就可以使用。CSS生成

① 老版本的Sass使用!来标识变量。改成$是多半因为!highlight-color看起来太丑了。

时，变量会被它们的值所替代。之后，如果你需要一个不同的值，只需要改变这个变量的值，则所有引用此变量的地方生成的值都会随之改变。

```
$highlight-color: #abcdef;

.selected {
  border: 1px $highlight-color solid;
}
```

看上边示例中的`$highlight-color`变量，它被直接赋值给`border`属性，当这段代码被编译输出CSS时，`$highlight-color`会被`#abcdef`这一颜色值所替代。产生的效果就是给`selected`这个类一条1像素宽、实心且颜色值为`#abcdef`的边框。

在声明变量时，变量值也可以引用其他变量。当你通过粒度区分，为不同的值取不同名字时，这相当有用。下例在独立的颜色值粒度上定义了一个变量，且在另一个更复杂的边框值粒度上也定义了一个变量：

```
$highlight-color: #abcdef;
$highlight-border: 1px $highlight-color solid;

.selected {
  border: $highlight-border;
}
```

这里，`$highlight-border`变量的声明中使用了`$highlight-color`这个变量。产生的效果就跟你直接为`border`属性设置了一个`1px $highlight-color solid`的值是一样的。

最后，我们来了解一下变量命名的实用技巧，以结束关于变量的介绍。

2.1.3　变量名用中划线还是下划线分隔

Sass的变量名可以与CSS中的属性名和选择器名称相同，包括中划线和下划线。这完全取决于个人的喜好，有些人喜欢使用中划线来分隔变量中的多个词（如`$highlight-color`），而有些人喜欢使用下划线（如`$highlight_color`）。使用中划线的方式更为普遍，这也是Compass和本书都用的方式。

不过，Sass并不想强迫任何人一定使用中划线或下划线，所以这两种用法相互兼容。用中划线声明的变量可以使用下划线的方式引用，反之亦然。这意味着即使Compass选择用中划线的命名方式，这并不影响你在使用Compass的样式中用下划线的命名方式进行引用：

```
$link-color: blue;

a {
  color: $link_color;
}
```

在上例中，`$link-color`和`$link_color`其实指向的是同一个变量。实际上，在Sass的大多数地方，中划线命名的内容和下划线命名的内容是互通的，除了变量，也包括对混合器（参见2.5节）和Sass函数（参见11.3节）的命名。但是在Sass中纯CSS部分不互通，比如类名、ID或属性名。

尽管变量自身提供了很多有用的地方，但是Sass基于变量提供的更为强大的工具才是我们关注的焦点。只有当变量与Sass的其他特性一起使用时，才能发挥其全部的潜能。接下来，我们将探讨其中一个非常重要的特性，即规则嵌套。

2.2 嵌套 CSS 规则

CSS中重复写选择器是非常恼人的。如果要写一大串指向页面中同一块的样式时，往往需要一遍又一遍地写同一个ID：

```
#content article h1 { color: #333 }
#content article p { margin-bottom: 1.4em }
#content aside { background-color: #eee }
```

像这种情况，Sass可以让你只写一遍，且使样式可读性更高。在Sass中，你可以像俄罗斯套娃那样在规则块中嵌套规则块。Sass在输出CSS时会帮你把这些嵌套规则处理好，避免你的重复书写。

```
#content {
  article {
    h1 { color: #333 }
    p { margin-bottom: 1.4em }
  }
  aside { background-color: #eee }
}
```

上边的例子，Sass会在输出CSS时把它转换成跟你之前看到的一样的效果。这个过程中，Sass用了两步，每一步都是像打开俄罗斯套娃那样把里边的嵌套规则块一个个打开。首先，把`#content`（父级）这个id放到`article`选择器（子级）和`aside`选择器（子级）的前边：

```
#content article {
  h1 { color: #333 }
  p { margin-bottom: 1.4em }
}
#content aside { background-color: #eee }
```

然后，`#content article`里边还有嵌套的规则，Sass重复一遍上边的步骤，把新的选择器添加到内嵌的选择器前边：

```
#content article h1 { color: #333 }
#content article p { margin-bottom: 1.4em }
#content aside { background-color: #eee }
```

一个给定的规则块，既可以像普通的CSS那样包含属性，又可以嵌套其他规则块。当你同时要为一个容器元素及其子元素编写特定样式时，这种能力就非常有用了。

```
#content {
  background-color: #f5f5f5;

  aside { background-color: #eee }
}
```

容器元素的样式规则会被单独抽离出来，而嵌套元素的样式规则会像容器元素没有包含任何属性时那样被抽离出来。

```
#content { background-color: #f5f5f5 }
#content aside { background-color: #eee }
```

大多数情况下这种简单的嵌套都没问题，但是有些场景下不行，比如你想要在嵌套的选择器里边立刻应用一个类似于：hover的伪类。为了解决这种以及其他情况，Sass提供了一个特殊结构&。

2.2.1　父选择器的标识符&

一般情况下，Sass在解开一个嵌套规则时就会把父选择器（#content）通过一个空格连接到子选择器的前边（article和aside）形成（#content article和#content aside）。这种在CSS里边被称为后代选择器，因为它选择ID为content的元素内所有命中选择器article和aside的元素。但在有些情况下你却不会希望Sass使用这种后代选择器的方式生成这种连接。

最常见的一种情况是当你为链接之类的元素写：hover这种伪类时，你并不希望以后代选择器的方式连接。比如说，下面这种情况Sass就无法正常工作：

```
article a {
   color: blue;
   :hover { color: red }
}
```

这意味着color: red这条规则将会被应用到选择器article a :hover，article元素内链接的所有子元素在被hover时都会变成红色。这是不正确的！你想把这条规则应用到超链接自身，而后代选择器的方式无法帮你实现。

解决之道为使用一个特殊的Sass选择器，即父选择器。在使用嵌套规则时，父选择器能对于嵌套规则如何解开提供更好的控制。它就是一个简单的&符号，且可以放在任何一个选择器可出现的地方，比如h1放在哪，它就可以放在哪。

```
article a {
  color: blue;
  &:hover { color: red }
}
```

当包含父选择器标识符的嵌套规则被打开时，它不会像后代选择器那样进行拼接，而是&被父选择器直接替换：

```
article a { color: blue }
article a:hover { color: red }
```

在为父级选择器添加：hover等伪类时，这种方式非常有用。同时父选择器标识符还有另外一种用法，你可以在父选择器之前添加选择器。举例来说，当用户在使用IE浏览器时，你会通过JavaScript在<body>标签上添加一个ie的类名，为这种情况编写特殊的样式如下：

```
#content aside {
```

```
  color: red;

  body.ie & { color: green }
}
```

Sass在选择器嵌套上是非常智能的，即使是带有父选择器的情况。当Sass遇到群组选择器（由多个逗号分隔开的选择器形成）也能完美地处理这种嵌套。

2.2.2 群组选择器的嵌套

在CSS里边，选择器h1、h2和h3会同时命中h1元素、h2元素和h3元素。与此类似，.button、button会命中button元素和类名为.button的元素。这种选择器称为*群组选择器*。群组选择器的规则会对命中群组中任何一个选择器的元素生效。

```
.button, button {
  margin: 0;
}
```

当看到上边这段代码时，你可能还没意识到会有重复性的工作。但会很快发现：如果你需要在一个特定的容器元素内对这样一个群组选择器进行修饰，情况就不同了。CSS的写法会让你在群组选择器中的每一个选择器前都重复一遍容器元素的选择器。

```
.container h1, .container h2, .container h3 { margin-bottom: .8em }
```

非常幸运，Sass的嵌套特性在这种场景下也非常有用。当Sass解开一个群组选择器规则内嵌的规则时，它会把每一个内嵌选择器的规则都正确地解出来：

```
.container {
  h1, h2, h3 {margin-bottom: .8em}
}
```

首先Sass将.container和h1、.container和h2、.container和h3分别组合，然后将三者重新组合成一个群组选择器，生成你前边看到的普通CSS样式。对于内嵌在群组选择器内的嵌套规则，处理方式也一样：

```
nav, aside {
  a {color: blue}
}
```

首先Sass将nav和a、aside和a分别组合，然后将二者重新组合成一个群组选择器：

```
nav a, aside a {color: blue}
```

处理这种群组选择器规则嵌套上的强大能力，正是Sass在减少重复敲写方面的贡献之一。尤其在当嵌套级别达到两层甚至三层以上时，与普通的CSS编写方式相比，只写一遍群组选择器大大减少了工作量。

有利必有弊，你需要特别注意群组选择器的规则嵌套生成的CSS。虽然Sass让你的样式表看上去很小，但实际生成的CSS却可能非常大，这会降低网站的速度。

关于选择器嵌套的最后一个方面，我们看看Sass如何处理*组合选择器*，比如>、+和~的使用。

你将看到，这种场景下你甚至无需使用父选择器标识符。

2.2.3　子组合选择器和同层组合选择器：>、+和~

上边这三个组合选择器必须和其他选择器配合使用，以指定浏览器仅选择某种特定上下文中的元素。

```
article section   { margin: 5px }
article > section { border: 1px solid #ccc }
```

你可以用子组合选择器>选择一个元素的直接子元素。上例中，第一个选择器会选择article下的所有命中section选择器的元素。第二个选择器只会选择article下紧跟着的子元素中命中section选择器的元素。

在下例中，你可以用同层相邻组合选择器+选择header元素后紧跟的p元素：

```
header + p { font-size: 1.1em }
```

你也可以用同层全体组合选择器~，选择所有跟在article后的同层article元素，不管它们之间隔了多少其他元素：

```
article ~ article { border-top: 1px dashed #ccc }
```

这些组合选择器可以毫不费力地应用到Sass的规则嵌套中。可以把它们放在外层选择器后边，或里层选择器前边：

```
article {
  ~ article { border-top: 1px dashed #ccc }
   > section  { background: #eee }
  dl > {
    dt { color: #333 }
    dd { color: #555 }
  }
  nav + & { margin-top: 0 }
}
```

Sass会如你所愿地将这些嵌套规则一一解开组合在一起：

```
article ~ article { border-top: 1px dashed #ccc }
article > footer  { background: #eee }
article dl > dt { color: #333 }
article dl > dd { color: #555 }
nav + article { margin-top: 0 }
```

在Sass中，不仅仅CSS规则可以嵌套，对属性进行嵌套也可以减少很多重复性的工作。

2.2.4　嵌套属性

在Sass中，除了CSS选择器，属性也可以进行嵌套。尽管编写属性涉及的重复不像编写选择器那么糟糕，但是要反复写border-style、border-width、border-color以及border-*等也是非常烦人的。在Sass中，你只需敲写一遍border：

```
nav {
  border: {
    style: solid;
    width: 1px;
    color: #ccc;
  }
}
```

嵌套属性的规则是这样的：把属性名从中划线-的地方断开，在根属性后边添加一个冒号:，紧跟一个{ }块，把子属性部分写在这个{ }块中。就像CSS选择器嵌套一样，Sass会把你的子属性一一解开，把根属性和子属性部分通过中划线-连接起来，最后生成的效果与你手动一遍遍写的CSS样式一样：

```
nav {
  border-style: solid;
  border-width: 1px;
  border-color: #ccc;
}
```

对于属性的缩写形式，你甚至可以像下边这样来嵌套，指明例外规则：

```
nav {
  border: 1px solid #ccc {
    left: 0px;
    right: 0px;
  }
}
```

这比下边这种同等样式的写法要好：

```
nav {
  border: 1px solid #ccc;
  border-left: 0px;
  border-right: 0px;
}
```

属性和选择器嵌套是非常伟大的特性，因为它们不仅大大减少了你的编写量，而且通过视觉上的缩进使你编写的样式结构更加清晰，更易于阅读和开发。

即便如此，随着你的样式表变得越来越大，这种写法也很难保持结构清晰。有时，处理这种大量样式的唯一方法就是把它们分拆到多个文件中。Sass通过对CSS原有`@import`规则的改进直接支持了这一特性。

2.3 导入 Sass 文件

CSS有一个特别不常用的特性，即`@import`规则，它允许在一个CSS文件中导入其他CSS文件。然而，后果是只有执行到`@import`时，浏览器才会去下载其他CSS文件，这导致页面加载起来特别慢。

Sass也有一个`@import`规则，但不同的是，Sass的`@import`规则在生成CSS文件时就把相关文件导入进来。这意味着所有相关的样式被归纳到了同一个CSS文件中，而无需发起额外的下载

请求。另外，所有在被导入文件中定义的变量和混合器（参见2.5节）均可在导入文件中使用。

使用Sass的`@import`规则并不需要指明被导入文件的全名。你可以省略.sass或.scss文件后缀（参见图2-1）。这样，在不修改样式表的前提下，你完全可以随意修改你或别人写的被导入的Sass样式文件语法，在Sass和Scss语法之间随意切换。举例来说，`@import"sidebar";`这条命令将把sidebar.scss文件中所有样式添加到当前样式表中。

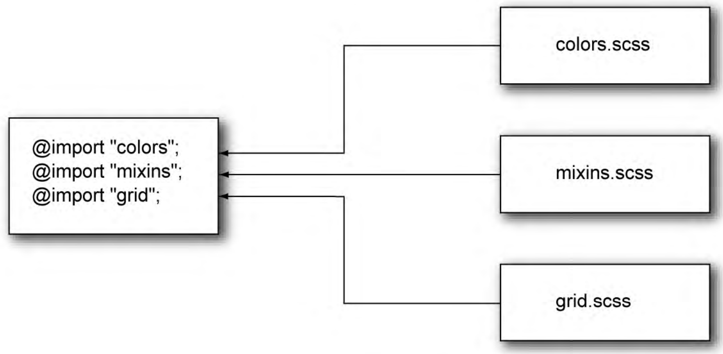


图2-1　导入Sass文件

本节将介绍如何使用Sass的`@import`来处理多个Sass文件。首先，我们将学习编写那些被导入的Sass文件，因为在一个大型Sass项目中，这样的文件是你最常编写的那一类。接着，了解集中导入Sass文件的方法，使你的样式可重用性更高，包括声明可自定义的变量值，以及在某一个选择器范围内导入Sass文件。最后，介绍如何在Sass中使用CSS原生的`@import`命令。

通常，有些Sass文件用于导入，你并不希望为每个这样的文件单独地生成一个CSS文件。对此，Sass用一个特殊的约定来解决。

2.3.1　使用Sass部分文件

当通过`@import`把Sass样式分散到多个文件时，你通常只想生成少数几个CSS文件。那些专门为`@import`命令而编写的Sass文件，并不需要生成对应的独立CSS文件，这样的Sass文件称为*局部文件*。对此，Sass有一个特殊的约定来命名这些文件。

此约定即，Sass局部文件的文件名以下划线开头。这样，Sass就不会在编译时单独编译这个文件输出CSS，而只把这个文件用作导入。当你`@import`一个局部文件时，还可以不写文件的全名，即省略文件名开头的下划线。举例来说，你想导入themes/_night-sky.scss这个局部文件里的变量，你只需在样式表中写`@import "themes/night-sky";`。

局部文件可以被多个不同的文件引用。当一些样式需要在多个页面甚至多个项目中使用时，这非常有用。在这种情况下，有时需要在你的样式表中对导入的样式稍作修改，Sass有一个功能刚好可以解决这个问题，即默认变量值。

2.3.2　默认变量值

一般情况下，你反复声明一个变量，只有最后一处声明有效且它会覆盖前边的值。举例说明：

```
$link-color: blue;
$link-color: red;

a {
  color: $link-color;
}
```

在上边的例子中，超链接的color会被设置为red。这可能并不是你想要的结果，假如你写了一个可被他人通过@import导入的Sass库文件，你可能希望导入者可以定制修改Sass库文件中的某些值。使用Sass的!default标签可以实现这个目的。它很像CSS属性中!important标签的对立面，不同的是!default用于变量，含义是：*如果这个变量被声明赋值了，那就用它声明的值，否则就用这个默认值。*

```
$fancybox-width: 400px !default;

.fancybox {
  width: $fancybox-width;
}
```

在上例中，如果用户在导入你的Sass局部文件之前声明了一个$fancybox-width变量，那么你的局部文件中对$fancybox-width赋值400px的操作就无效。如果用户没有做这样的声明，则$fancybox-width将默认为400px。

接下来我们将学习嵌套导入，它允许只在某一个选择器的范围内导入Sass局部文件。

2.3.3　嵌套导入

跟原生的CSS不同，Sass允许@import命令写在CSS规则内。这种导入方式下，生成对应的CSS文件时，局部文件会被直接插入到CSS规则内导入它的地方。举例说明，有一个名为_blue-theme.scss的局部文件，内容如下：

```
aside {
  background: blue;
  color: white;
}
```

然后把它导入到一个CSS规则内，如下所示：

```
.blue-theme {@import "blue-theme"}
```

生成的结果跟你直接在.blue-theme选择器内写_blue-theme.scss文件的内容完全一样。

```
.blue-theme {
  aside {
    background: blue;
    color: #fff;
  }
}
```

被导入的局部文件中定义的所有变量和混合器（参见2.5节），也会在这个规则范围内生效。这些变量和混合器不会全局有效，这样我们就可以通过嵌套导入只对站点中某一特定区域运用某种颜色主题或其他通过变量配置的样式。

有时，可用CSS原生的`@import`机制，在浏览器中下载必需的CSS文件。Sass也提供了几种方法来达成这种需求。

2.3.4　原生的CSS导入

由于Sass兼容原生的CSS，所以它也支持原生的CSS`@import`。尽管通常在Sass中使用`@import`时，Sass会尝试找到对应的Sass文件并导入进来，但在下列三种情况下会生成原生的CSS`@import`，尽管这会造成浏览器解析CSS时的额外下载：

- 被导入文件的名字以.css结尾；
- 被导入文件的名字是一个URL地址（比如http://sass-lang.com/stylesheets/application.css），由此可用谷歌字体API提供的相应服务；
- 被导入文件的名字是CSS的`url()`值。

这就是说，你不能用Sass的`@import`直接导入一个原始的CSS文件，因为Sass会认为你想用CSS原生的`@import`。但是，因为Sass的语法完全兼容CSS，所以你可以把原始的CSS文件改名为.scss后缀，即可直接导入了。

文件导入是保证Sass的代码可维护性和可读性的重要一环。次之但亦非常重要的就是注释了。注释可以帮助样式作者记录写Sass的过程中的想法。在原生的CSS中，注释对于其他人是直接可见的，但Sass提供了一种方式可在生成的CSS文件中按需抹掉相应的注释。

2.4　静默注释

CSS中注释的作用包括帮助你组织样式、以后你看自己的代码时明白为什么这样写，以及简单的样式说明。但是，你并不希望每个浏览网站源码的人都能看到所有注释。

Sass另外提供了一种不同于CSS标准注释格式`/* ... */`的注释语法，即静默注释，其内容不会出现在生成的CSS文件中。静默注释的语法跟JavaScript、Java等类C的语言中单行注释的语法相同，它们以`//`开头，注释内容直到行末。

```
body {
  color: #333; // 这种注释内容不会出现在生成的css文件中
  padding: 0; /* 这种注释内容会出现在生成的css文件中 */
}
```

实际上，CSS的标准注释格式`/* ... */`内的注释内容亦可在生成的CSS文件中抹去。当注释出现在原生CSS不允许的地方，如在CSS属性或选择器中，Sass将不知如何将其生成到对应CSS文件中的相应位置，于是这些注释被抹掉。

```
body {
  color /* 这块注释内容不会出现在生成的css中 */: #333;
```

```
  padding: 1em; /* 这块注释内容也不会出现在生成的css中 */ 0;
}
```

你已经掌握了Sass的静默注释，了解了保持Sass条理性和可读性的最基本的三个方法：嵌套、导入和注释。现在，我们要进一步学习新特性，这样我们不但能保持条理性还能写出更好的样式。首先要介绍的内容是：使用混合器抽象你的相关样式。

2.5　混合器

如果你的整个网站中有几处小小的样式类似（例如一致的颜色和字体），那么使用变量来统一处理这种情况是非常不错的选择。但是当你的样式变得越来越复杂，你需要大段大段的重用样式的代码，独立的变量就没办法应付这种情况了。你可以通过Sass的混合器实现大段样式的重用。

混合器使用`@mixin`标识符定义。看上去很像其他的CSS `@`标识符，比如说`@media`或者CSS3的`@font-face`。这个标识符给一大段样式赋予一个名字，这样你就可以轻易地通过引用这个名字重用这段样式。下边的这段Sass代码，定义了一个非常简单的混合器，目的是添加跨浏览器的圆角边框。

```
@mixin rounded-corners {
  -moz-border-radius: 5px;
  -webkit-border-radius: 5px;
  border-radius: 5px;
}
```

然后就可以在你的样式表中通过`@include`来使用这个混合器，放在你希望的任何地方。`@include`调用会把混合器中的所有样式提取出来放在`@include`被调用的地方。如果像下边这样写：

```
.notice {
  background-color: green;
  border: 2px solid #00aa00;
  @include rounded-corners;
}
```

Sass最终生成：

```
.notice {
  background-color: green;
  border: 2px solid #00aa00;
  -moz-border-radius: 5px;
  -webkit-border-radius: 5px;
  border-radius: 5px;
}
```

在`.notice`中的属性`border-radius`、`-moz-border-radius`和`-webkit-border-radius`全部来自`rounded-corners`这个混合器。这一节将介绍使用混合器来避免重复。通过使用参数，你可以使用混合器把你样式中的通用样式抽离出来，然后轻松地在其他地方重用。实际上，混合器太好用了，一不小心你可能会过度使用。大量的重用可能会导致生成的样式表过大，导致加载缓慢。所以，首先我们将讨论混合器的使用场景，避免滥用。

2.5.1 何时使用混合器

利用混合器，可以很容易地在样式表的不同地方共享样式。如果你发现自己在不停地重复一段样式，那就应该把这段样式构造成优良的混合器，尤其是这段样式本身就是一个逻辑单元，比如说是一组放在一起有意义的属性。

判断一组属性是否应该组合成一个混合器，一条经验法则就是你能否为这个混合器想出一个好的名字。如果你能找到一个很好的短名字来描述这些属性修饰的样式，比如`rounded-corners`、`fancy-font`或者`no-bullets`，那么往往能够构造一个合适的混合器。如果你找不到，这时候构造一个混合器可能并不合适。

混合器在某些方面跟CSS类很像。都是让你给一大段样式命名，所以在选择使用哪个的时候可能会产生疑惑。最重要的区别就是类名是在HTML文件中应用的，而混合器是在样式表中应用的。这就意味着类名具有语义化含义，而不仅仅是一种*展示性的描述*：用来描述HTML元素的含义而不是HTML元素的外观。而另一方面，混合器是展示性的描述，用来描述一条CSS规则应用之后会产生怎样的效果。

在之前的例子中，`.notice`是一个有语义的类名。如果一个HTML元素有一个`notice`的类名，就表明了这个HTML元素的用途：向用户展示提醒信息。`rounded-corners`混合器是展示性的，它描述了包含它的CSS规则最终的视觉样式，尤其是边框角的视觉样式。

混合器和类配合使用写出整洁的HTML和CSS，因为使用语义化的类名亦可以帮你避免重复使用混合器。为了保持你的HTML和CSS的易读性和可维护性，在写样式的过程中一定要铭记二者的区别。

有时候仅仅把属性放在混合器中还远远不够，可喜的是，Sass同样允许你把CSS规则放在混合器中。

2.5.2 混合器中的CSS规则

混合器中不仅可以包含属性，也可以包含CSS规则，包含选择器和选择器中的属性，如代码清单2-1所示。

代码清单2-1 包含CSS规则的选择器

```
@mixin no-bullets {
  list-style: none;
  li {
    list-style-image: none;
    list-style-type: none;
    margin-left: 0px;
  }
}
```

当一个包含CSS规则的混合器通过`@include`包含在一个父规则中时，在混合器中的规则最终会生成父规则中的嵌套规则。举个例子，看看下边的Sass代码，这个例子中使用了`no-bullets`这个混合器：

```
ul.plain {
  color: #444;
  @include no-bullets;
}
```

Sass的`@include`指令会将引入混合器的那行代码替换成混合器里边的内容。最终，上边的例子如代码清单2-2所示。

代码清单2-2 在`ul.plain`中包含了`no-bullets`的最终代码

```
ul.plain {
  color: #444;
  list-style: none;
}
ul.plain li {
  list-style-image: none;
  list-style-type: none;
  margin-left: 0px;
}
```

混合器中的规则甚至可以使用Sass的父选择器标识符`&`。使用起来跟不用混合器时一样，Sass解开嵌套规则时，用父规则中的选择器替代`&`。

如果一个混合器只包含CSS规则，不包含属性，那么这个混合器就可以在文档的顶部调用，写在所有的CSS规则之外。如果你只是为自己写一些混合器，这并没有什么大的用途，但是当你使用一个类似于Compass的库时，你会发现，这是提供样式的好方法，原因在于你可以选择是否使用这些样式。

接下来你将学习如何通过给混合器传参数来让混合器变得更加灵活和可重用。

2.5.3 给混合器传参

混合器并不一定总得生成相同的样式。可以通过在`@include`混合器时给混合器传参，来定制混合器生成的精确样式。当`@include`混合器时，参数其实就是可以赋值给CSS属性值的变量。如果你写过JavaScript，这种方式跟JavaScript的`function`很像：

```
@mixin link-colors($normal, $hover, $visited) {
  color: $normal;
  &:hover { color: $hover; }
  &:visited { color: $visited; }
}
```

当混合器被@include时，你可以把它当作一个CSS函数来传参。如果你像下边这样写：

```
a {
  @include link-colors(blue, red, green);
}
```

Sass最终生成的是：

```
a { color: blue; }
a:hover { color: red; }
a:visited { color: green; }
```

当你@include混合器时，有时候可能会很难区分每个参数是什么意思，参数之间是一个什么样的顺序。为了解决这个问题，Sass允许通过语法$name: value的形式指定每个参数的值。这种形式的传参，参数顺序就不必再在乎了，只需要保证没有漏掉参数即可：

```
a {
  @include link-colors(
    $normal: blue,
    $visited: green,
    $hover: red
  );
}
```

尽管给混合器加参数来实现定制很好，但是有时有些参数我们没有定制的需要，这时候也需要赋值一个变量就变成很痛苦的事情了。所以Sass允许混合器声明时给参数赋默认值。

2.5.4　默认参数值

为了在@include混合器时不必传入所有的参数，我们可以给参数指定一个默认值。参数默认值使用$name: default-value的声明形式，默认值可以是任何有效的CSS属性值，甚至是其他参数的引用，如代码清单2-3所示。

代码清单2-3　设置参数默认值

```
@mixin link-colors(
  $normal,
  $hover: $normal,
  $visited: $normal
  ) {
  color: $normal;
  &:hover { color: $hover; }
  &:visited { color: $visited; }
}
```

如果像下边这样调用：@include link-colors(red)， $hover和$visited也会被自动赋值为red。

混合器只是Sass样式重用特性中的一个。我们已经了解到混合器主要用于样式展示层的重用，如果你想重用语义化的类呢？这就涉及Sass的另一个重要的重用特性：选择器继承。

2.6　使用选择器继承来精简 CSS

使用Sass的时候，最后一个减少重复的主要特性就是选择器继承。基于Nicole Sullivan面向对象的CSS的理念，选择器继承是说一个选择器可以继承为另一个选择器定义的所有样式。这个通过@extend语法实现，如代码清单2-4所示。

代码清单2-4　通过选择器继承继承样式

```
.error {
```

```
  border: 1px red;
  background-color: #fdd;
}
.seriousError {
   @extend .error;
   border-width: 3px;
}
```


在上边的代码中，.seriousError将会继承样式表中任何位置处为.error定义的所有样式。以 class="seriousError" 修饰的 HTML 元素最终的展示效果就好像是 class="seriousError error"。相关元素不仅会拥有一个3px宽的边框，而且这个边框将变成红色的，这个元素同时还会有一个浅红色的背景，因为这些都是在.error里边定义的样式。

.seriousError不仅会继承.error自身的所有样式,任何跟.error有关的组合选择器样式也会被.seriousError以组合选择器的形式继承，如代码清单2-5所示。

代码清单2-5 .seriousError从.error继承样式

```
.error a {                        <-- 应用到.seriousError a
  color: red;
  font-weight: bold;
}

h1.error {                        <-- 应用到h1.seriousError
  font-size: 1.3em;
}
```

如上所示，在class="seriousError"的HTML元素内的超链接也会变成红色和粗体。

本节将介绍与混合器相比，哪种情况下更适合用继承。接下来在探索继承的工作细节之前，我们先了解一下继承的高级用法。最后，我们将看看使用继承可能会有哪些坑，学习如何避免这些坑。

2.6.1 何时使用继承

2.5.1节介绍了混合器主要用于展示性样式的重用，而类名用于语义化样式的重用。因为继承是基于类的（有时是基于其他类型的选择器），所以继承应该是建立在语义化的关系上。当一个元素拥有的类（比如说.seriousError）表明它属于另一个类（比如说.error），这时使用继承再合适不过了。

这有点抽象，所以我们从几个方面来阐释一下。想象一下你正在编写一个页面，给HTML元素添加类名，你发现你的某个类（比如说.seriousError）另一个类（比如说.error）的细化。你会怎么做?

- 你可以为这两个类分别写相同的样式，但是如果有大量的重复怎么办？使用Sass时，我们提倡的就是不要做重复的工作。
- 你可以使用一个选择器组（比如说.error、.seriousError）给这两个选择器写相同的样式。如果.error的所有样式都在同一个地方，这种做法很好，但是如果是分散在样式表的不同地方呢？再这样做就困难多了。
- 你可以使用一个混合器为这两个类提供相同的样式，但当.error的样式修饰遍布样式表

中各处时，这种做法面临着跟使用选择器组一样的问题。这两个类也不是恰好有相同的样式。你应该更清晰地表达这种关系。

- 综上所述你应该使用`@extend`。让`.seriousError`从`.error`继承样式，使两者之间的关系非常清晰。更重要的是无论你在样式表的哪里使用`.error`，`.seriousError`都会继承其中的样式。

现在你已经更好地掌握了何时使用继承，以及继承有哪些突出的优点，接下来我们看看一些高级用法。

2.6.2　继承的高级用法

任何CSS规则都可以继承其他规则，几乎任何CSS规则也都可以被继承。大多数情况你可能只想对类使用继承，但是有些场合你可能想做得更多。最常用的一种高级用法是继承一个HTML元素的样式。尽管默认的浏览器样式不会被继承，因为它们不属于样式表中的样式，但是你对HTML元素添加的所有样式都会被继承[①]。

接下来的这段代码定义了一个名为`disabled`的类，样式修饰使它看上去像一个灰掉的超链接。通过继承a这一超链接元素来实现：

```
.disabled {
  color: gray;
  @extend a;
}
```

假如一条样式规则继承了一个复杂的选择器，那么它只会继承这个复杂选择器命中的元素所应用的样式。举例来说，如果`.seriousError @extend .important.error`，那么`.important.error`和`h1.important.error`的样式都会被`.seriousError`继承，但是`.important`或者`.error`下的样式则不会被继承。这种情况下你很可能希望`.seriousError`能够分别继承`.important`或者`.error`下的样式。

如果一个选择器序列（`#main .seriousError`）`@extend`另一个选择器（`.error`），那么只有完全命中`#main .seriousError`这个选择器的元素才会继承`.error`的样式，就像单个类名继承那样。拥有`class="seriousError"`的`.main`元素之外的元素不会受到影响。

像`#main .error`这种选择器序列是不能被继承的。这是因为从`#main .error`中继承的样式一般情况下会跟直接从`.error`中继承的样式基本一致，细微的区别往往使人迷惑。

现在你已经了解了通过继承能够做些什么事情，接下来我们将学习继承的工作细节，在生成对应CSS的时候，Sass具体干了些什么事情。

2.6.3　继承的工作细节

跟变量和混合器不同，继承不是仅仅用CSS样式替换`@extend`处的代码那么简单。为了不让

① 如果你使用了CSS重置就存在争议了，因为浏览器默认为这个元素添加的样式都会被你的样式表覆盖。

你对生成的CSS感觉奇怪，对这背后的工作原理有一定了解是非常重要的。

@extend背后最基本的想法是，如果.seriousError @extend .error，那么样式表中的任何一处.error都用.error、.seriousError这一选择器组进行替换。这就意味着相关样式会如预期那样应用到.error和.seriousError。当.error出现在复杂的选择器中，比如说h1.error、.error a或者#main .sidebar input.error[type="text"]，那情况就变得复杂多了，但是不用担心，Sass已经为你考虑到了这些。

关于@extend有两个要点你应该知道。

- 跟混合器相比，继承生成的CSS代码相对更少。因为继承仅仅是重复选择器，而不会重复属性，所以使用继承往往比混合器生成的CSS体积更小。如果你非常关心你站点的速度，请牢记这一点。
- 继承遵从CSS层叠的规则。当两个不同的CSS规则应用到同一个HTML元素上时，并且这两个不同的CSS规则对同一属性的修饰存在不同的值，CSS层叠规则会决定应用哪个样式。相当直观：通常权重更高的选择器胜出，如果权重相同，定义在后边的规则胜出。

混合器本身不会引起CSS层叠的问题，因为混合器把样式直接放到了CSS规则中，而继承存在样式层叠的问题。被继承的样式会保持原有定义位置和选择器权重不变。通常来说这并不会引起什么问题，但是知道这点总没有坏处。

2.6.4 使用继承的最佳实践

通常使用继承会让你的CSS美观、整洁。因为继承只会在生成CSS时复制选择器，而不会复制大段的CSS属性。但是如果你不小心，可能会让生成的CSS中包含大量的选择器复制。

避免这种情况出现的最好方法就是不要在CSS规则中使用后代选择器（比如.foo .bar）去继承CSS规则。如果你这么做，同时被继承的CSS规则有通过后代选择器修饰的样式，生成CSS中的选择器的数量很快就会失控：

```
.foo .bar { @extend .baz; }

.bip .baz { a: b; }
```

在上边的例子中，Sass必须保证应用到.baz的样式同时也要应用到.foo .bar（位于class="foo"的元素内的class="bar"的元素）。例子中有一条应用到.bip .baz（位于class="bip"的元素内的class="baz"的元素）的CSS规则。当这条规则应用到.foo .bar时，可能存在三种情况，如代码清单2-6所示。

代码清单2-6 继承可能迅速变复杂

```
<!-- Case 1 -->
<div class="foo">
  <div class="bip">
    <div class="bar">...</div>
  </div>
</div>
```

```
<!-- Case 2 -->
<div class="bip">
  <div class="foo">
    <div class="bar">...</div>
  </div>
</div>

<!-- Case 3 -->
<div class="foo bip">
  <div class="bar">...</div>
</div>
```

为了应付这些情况，Sass必须生成三种选择器组合（仅仅是`.bip .foo .bar`不能覆盖所有情况）。如果任何一条规则里边的后代选择器再长一点，Sass需要考虑的情况就会更多。实际上Sass并不总是会生成所有可能的选择器组合，即使是这样，选择器的个数依然可能会变得相当大，所以如果允许，尽可能避免这种用法。

值得一提的是，只要你想，你完全可以放心地继承有后代选择器修饰规则的选择器，不管后代选择器多长，但有一个前提就是，不要用后代选择器去继承。

2.7　小结

这章介绍了构成Sass和Compass的最基本部分。通过这章介绍的工具你可以轻松地使用Sass编写清晰、无冗余、语义化的CSS。对于Sass提供的工具你已经有了一个比较深入的了解，同时也掌握了何时使用这些工具的指导原则。

变量是Sass提供的最基本的工具。通过变量可以让独立的CSS值变得可重用，无论是在一条单独的规则范围内还是在整个样式表中。变量、混合器的命名甚至Sass的文件名，可以互换通用_和-。

同样基础的是Sass的嵌套机制。嵌套允许CSS规则内嵌套CSS规则，减少重复编写常用的选择器，同时让样式表的结构一眼望去更加清晰。Sass同时提供了特殊的父选择器标识符&，通过它可以构造出更高效的嵌套。

你也已经学到了Sass的另一个重要特性，样式导入。通过样式导入可以把分散在多个Sass文件中的内容合并生成到一个CSS文件，避免了项目中有大量的CSS文件通过原生的CSS `@import`带来的性能问题。通过嵌套导入和默认变量值，导入可以构建更强有力的、可定制的样式。

混合器允许用户编写语义化样式的同时避免视觉层面上样式的重复。你不仅学到了如何使用混合器减少重复，同时学习到了如何使用混合器让你的CSS变得更加可维护和语义化。

最后，我们学习了与混合器相辅相成的选择器继承。继承允许你声明类之间语义化的关系，通过这些关系可以保持你的CSS的整洁和可维护性。

如果你已经开始应用在这章学到的内容，那么你已经可以写出很好的Sass了，但是很快你会开始担忧是不是有其他方法可以做得更好。你会考虑如何使混合器变得更加可重用，如何让Sass帮你进行一些计算。你会考虑是否可以只控制一个颜色变量，就改变你整个站点的主题，以及是否可以让样式高度可重用，以便在你的站点以及你的朋友间随意分享。

下一章将探索学习Sass的脚本特性，通过这个特性你可以实现上边提到的一切。

Part 2 第二部分

在实战中使用 Sass 和 Compass

通过前两章的学习，你已经结识了Sass和Compass，并且了解了Sass语法的核心特性。接下来的3章里，你将看到Sass和Compass在实际应用中的价值，看到它们如何帮助你解决之前枯燥乏味的任务，助你通过更少量的工作写出更强有力的样式表。

第3章将介绍Compass如何使Web设计中最基础的部分——布局变得简单。在这一章，我们探讨网格布局背后的原理，以及Compass提供的能够使网格布局大大简化且更加灵活的工具。传统CSS网格布局系统要求你在HTML标签上写很多可表明样式的类名。在这一章你将学到如何在Compass中使用Blueprint和960网格布局系统，从而避免HTML标签类名的困扰。你将看到Sass的动态特性如何帮助你建立起一个网格布局框架，并且仅仅通过修改几个变量能就轻地改变它。最后展示如何运用Compass的排版辅助器处理垂直韵律。

第4章将深挖Compass的工具箱，看看Compass的混合器如何帮助你省去编写重复样表式的辛苦工作。Compass通过动态混合器封装了很多经典的样式类型。你将学习如何重置浏览器默认的样式，使老版本浏览器支持HTML5标签。我们还将看看可以修饰超链接和水平行内列表样式的混合器以及其他有用的排版和布局模式。

第5章展示了Compass如何使你免去编写跨浏览器的CSS3的痛苦。你会发现通过Compass实现盒阴影、边框圆角以及背景渐变等新锐样式是那么简单，而且还不需要关心厂商前缀以及各浏览器的实现细节。我们将看看Compass如何简化`@font-face`，甚至通过使用简单的CSS PIE整合，帮助你在旧版本的IE中支持一些CSS3特性。

当你完成这部分学习时，会对Compass如何融入你的样式表工作流程以及帮你解决日常问题有一个很好的理解。你将会对动态样式表的巨大能力以及如何更轻松地写出更好的样式表有一个真切的感受。在下一部分，我们将看看Compass的智能自动化CSS精灵，如何将样式表从开发上线到生产环境，以及如何优化样式表以获得更高性能。

第 3 章

无需计算玩转CSS网格布局

本章内容

- 网格布局的基本原理以及何时使用网格布局
- 使用Compass时的CSS网格布局框架选项
- 使用排版辅助器处理垂直韵律问题

3.1 网格布局介绍

留白在Web设计中非常有用，但却常常被忽略。留白就是你的布局中内容之间的空隙。留白通过在不同类型的信息间制造视觉隔离，帮助你更好地浏览内容，或者让你把注意力集中到更重要的元素上。

网格布局框架可以帮你在Web页面中高效地使用留白，对行列间的内容，以及行列间的间隙提供统一的尺寸。尽管从印刷机发明之日起，网格布局在印刷中就已经非常普遍了，但是在Web设计中只流行只有短短几年。除了能在你的设计中提供一些留白用法的最佳实践，CSS网格布局还可以迅速调整内容区域宽度，从而快速构建新的布局原型。

3.1.1 不使用CSS网格布局或者不使用网格辅助设计

统一的留白除了有美学上的考虑，还能帮你更好地浏览和阅读内容。我们的视线会被页面中元素之间的空白吸引，不均等的空白可以一下子抓住你的注意力。这样做有利有弊，有时太多焦点也会造成认知上的干扰。

同写在贺卡中的一段文本相比，写在横格纸上的文本段落会更易辨识。没有一个统一的基线，你的手书会变差，辨识度也会严重降低。同样的道理，脱离了网格的设计，留给你的将是不一致的尺寸和随意的对齐，这样会大大降低设计的冲击力。CSS网格布局框架建立了一个各种统一尺寸的规范，避免你陷入在布局中失去焦点的危险。接下来让我们看一个简单的网格布局示例。

3.1.2 网格布局系统或框架及其工作原理

为防你以前从未接触过网格布局，我们先来看一下一些实战中的网格布局。假如你现在能很

方便地上网，接下来你可能希望跟着一步步在线操作。我们将看一看Geoffrey Grosenbach的超棒的PeepCode博客（http://blog.peepcode.com/archives），如图3-1所示，PeepCode充分利用了CSS网格布局的优势。

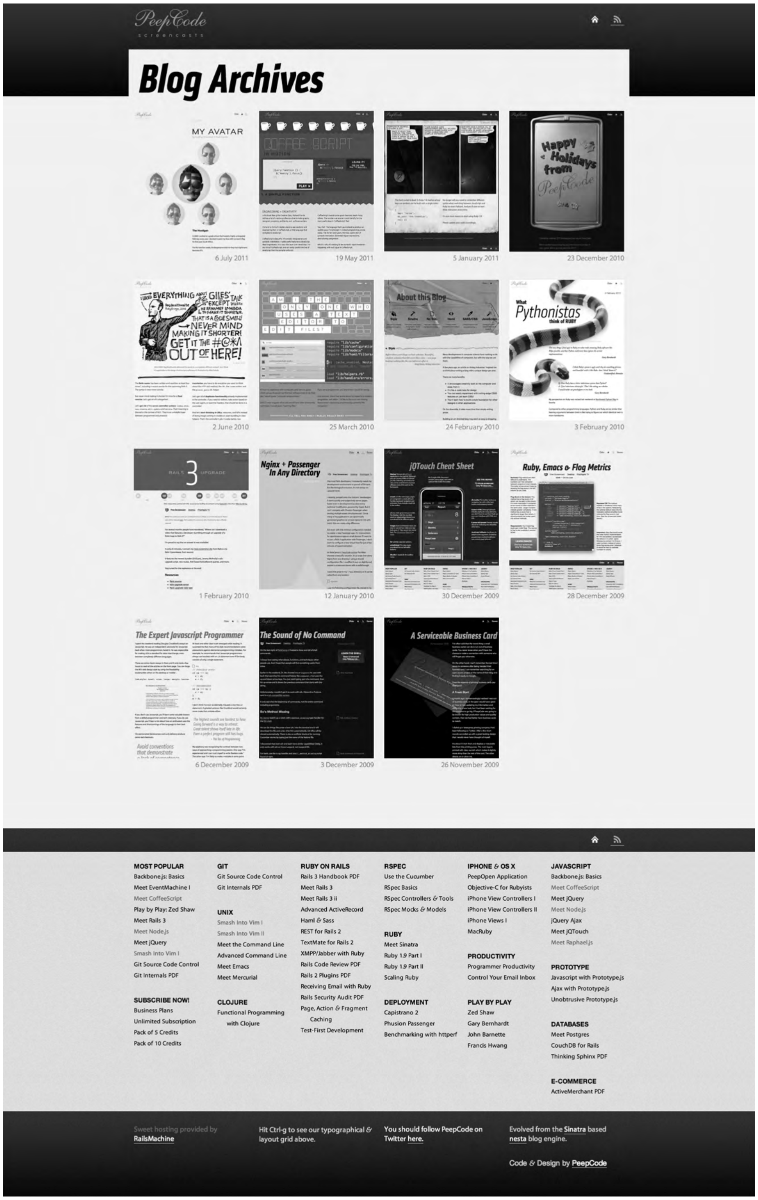


图3-1　使用CSS网格布局的PeepCode

看一下图3-1，在这个网格布局中一共有多少列？如果你说有4个，并不完全正确。它确实主要有4个大列，但是看一下页脚，下边有6列的链接。如果你按下Ctrl+G，你会看到Geoffrey在页面中藏了一个非常棒的复活节彩蛋，网格布局的底层网格显现出来了（如图3-2所示）。

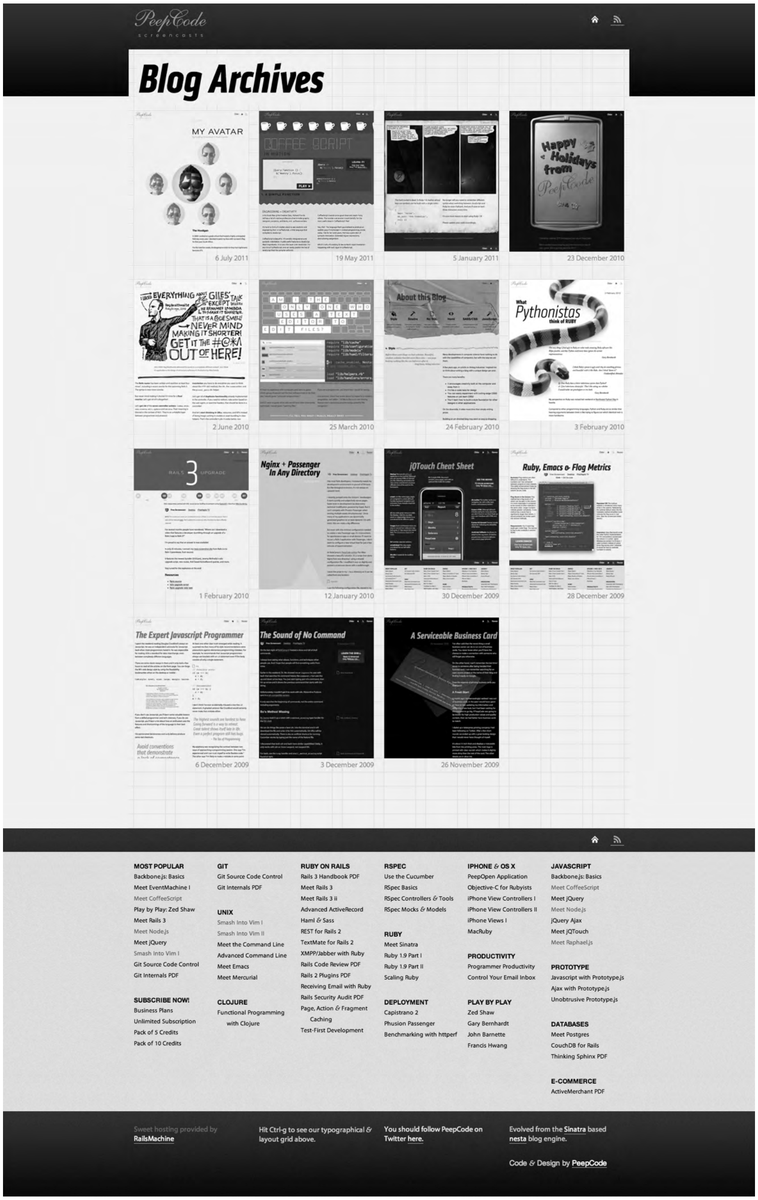


图3-2　显现的网格

如图3-2所示，页面实际上有12个列。每一个缩略图大列由3个小列构成（4×3=12），而在页脚中的每个大列由两个小列构成（6×2=12）。利用这种布局方式，我们能够轻松地以一种看上去很舒服的方式排布图片，当然了它的功能远不止于此。如果你一篇篇翻看Geoffrey存档中的文章，并且使用Ctrl+G快捷键，你会发现每一篇都有类似的网格布局（如图3-3所示）。

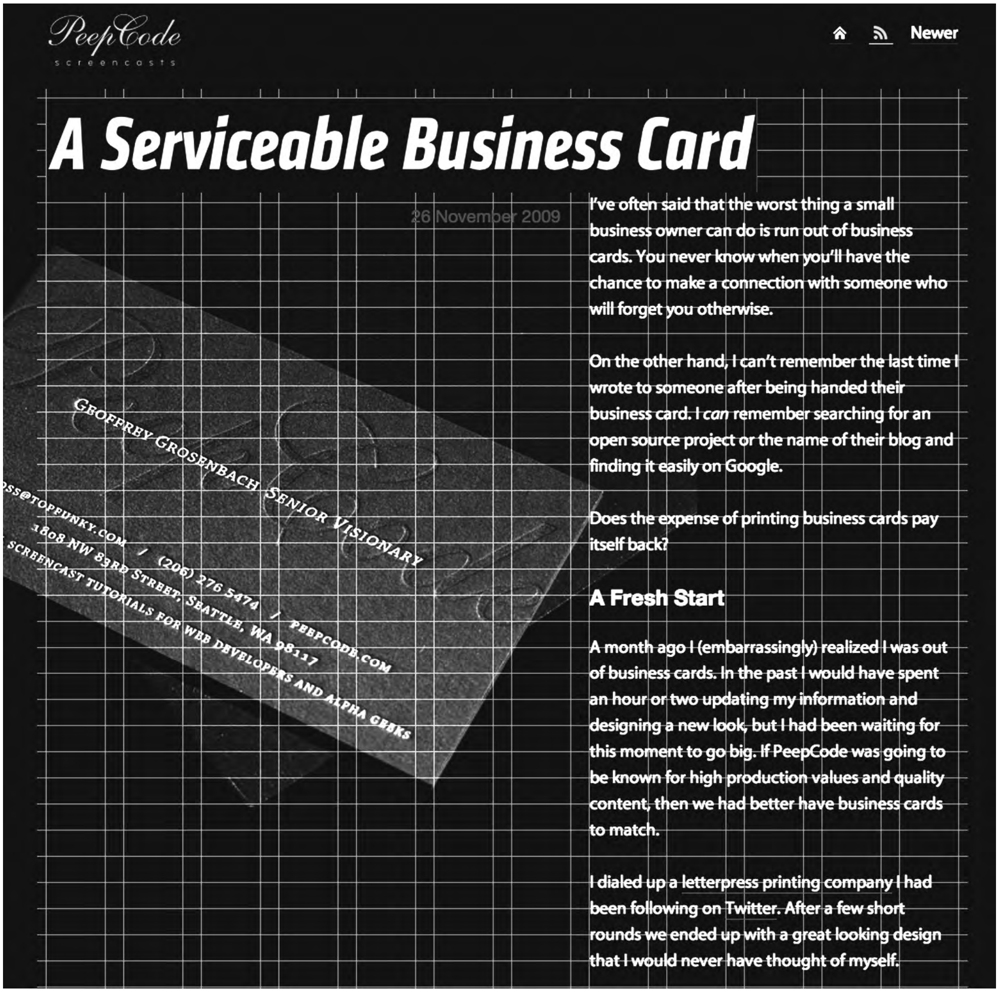


图3-3　适用于文本的网格布局

图3-3中的文章只有5列宽，这个布局使用了一个7列的留白以背景图的方式来展示一张名片，这张名片图用来支撑文章中的内容。

至此你已经看到了网格布局能够提供的巨大价值，Sass和Compass在这其中能起到什么作用呢？

3.1.3　使用Sass和Compass进行网格布局

网格布局的核心原理很简单，无非就是把内容和容器通过简单的数学计算进行均分，然后分配。Compass（和Sass）帮你处理这些计算，把你从通过类名来指定网格宽度的繁杂工作中解放出来。使用Sass中的变量，你可以轻易地配置网格样式，仅仅修改几个变量就能尝试另外一种设定。

接下来，我们看看网格系统由哪几部分构成，以及Sass和Compass如何帮你处理它们。

3.2　开始使用网格布局

这一章将介绍一些CSS网格布局的先进特性。如果你对CSS网格布局已经很熟悉了，这将是一个复习过程。如果这是你第一次接触CSS网格布局，这章将帮助你迅速入门。现在，让我们定义一些术语。

3.2.1　术语

尽管每种CSS网格布局框架都有一套它们自己的关于网格元素的命名，但是有些概念却是所有框架共享的（如表3-1所示）。

表3-1　网格布局框架术语

术语名	定　　义	是否涉及HTML标签
列	内容度量的垂直单位	否
容器	构成一个网格布局的HTML元素	是
槽	网格布局中列与列之间的统一留白	否

表3-1中的术语是CSS网格布局的核心，正如你所见，只有网格容器会出现在你的标签中。

1. 列

列是网格布局框架的最核心部分。在印刷界，如果内容为王，那么列就是隐藏在王座之后的强大力量。作者被称为*专栏作家*，分类广告以*列*和英寸为单位进行售卖。在Web设计中，我们早已迈过了使用红、蓝或粉色链接，居中排布的落后时代。但是相比印刷，我们在好多方面依然很落后。CSS很久之前就具有了调整水平内容渲染方式的能力，但是垂直布局对其一直是个挑战。原生支持基于列的布局才刚刚被CSS规范收录，所以还需要很多年你才能真正将其利用起来。这段真空期刚好被CSS网格布局框架和基于列的布局填充了。

再看一眼图3-2。那些垂直的阴影条就是列。你会发现它们每个都有30px宽，同时带有统一的留白。有很多技术可以在CSS中实现列的效果。在3.2节中我们将看几个网格布局系统。到现在为止，请牢记所有的CSS网格布局都有列的概念，并且这些列在同一容器内是等宽的。下面我们将探究一下容器。

2. 容器

回顾图3-2中Blueprint的例子，你应该会有这样一个印象：CSS网格布局把整个页面变成了基

于列的布局，很像一张报纸。在Web中，用户的屏幕大小和分辨率是你不能控制的，所以你并不知道页面在用户那里实际会有多大。在CSS网格布局中，你在一个容器内构造一个基于列的布局。一个网格布局既可以只有一个容器也可以有多个容器。有时候，你或许会构造不同列宽和列数的容器，我们后面介绍的960网格布局系统中就会有这种情况。在CSS网格框架中，一个容器一般说来就是一个封装元素，多数时候是一个DIV给它指定一个用来实现网格布局的CSS选择器。

现在你已经知道CSS网格布局都是通过一个或多个容器来容纳列的，接下来让我们看看网格布局的另一个关键点——槽。

3. 槽

就像房顶上的槽用来收纳雨水然后将其从房顶引流到排水沟一样，槽帮助我们的眼睛有效地注意到内容块之间的边界。再次回顾一下图3-2中的例子。阴影列之间的缺口就是槽。值得注意的是，就像列一样，槽也是有统一宽度的，在图3-2中宽度是10px。不同的网格布局会使用不同的算法来满足列式布局的需要，但是全部是基于列数、列宽和槽宽的。

到目前为止，我们只看到了基于像素的、固定宽度的列和槽。下一节一起看看有哪些可选的网格布局。

3.2.2 是否使用网格布局，要语义还是要实用

任何技术都不乏批评者。CSS网格布局框架当然也不例外。批评人士抱怨说：使用CSS类名来指定网格布局是把展现和内容耦合了。在这个语义阵营的人坚称，标签应该只是关于内容和数据的，不应该包含任何关于展现的信息。而实用主义者则认为能影响语义的是标签而不是类名。

幸好，Compass提供了两种选择，在不修改原有类名的情况下，你既可以通过添加类名又可以通过使用混合器来实现网格布局。

3.2.3 固定的网格布局还是流动的网格布局

由于网络用户的屏幕尺寸不一，设计人员有两种选择：要么选择一种对于大多数用户合理的固定布局大小（并且限制这种布局内的内容不溢出），要么实现一种灵活的流动布局，让内容自适应用户的屏幕，甚至当浏览器窗口大小改变时也会自适应。

图3-4是Stephen Bau基于Nathan Smith著名的960网格系统编写的一个流动网格布局示例。在对比图中你可以看到同样的16列的网格布局，当用户的浏览器窗口变大时，列宽也会扩大。

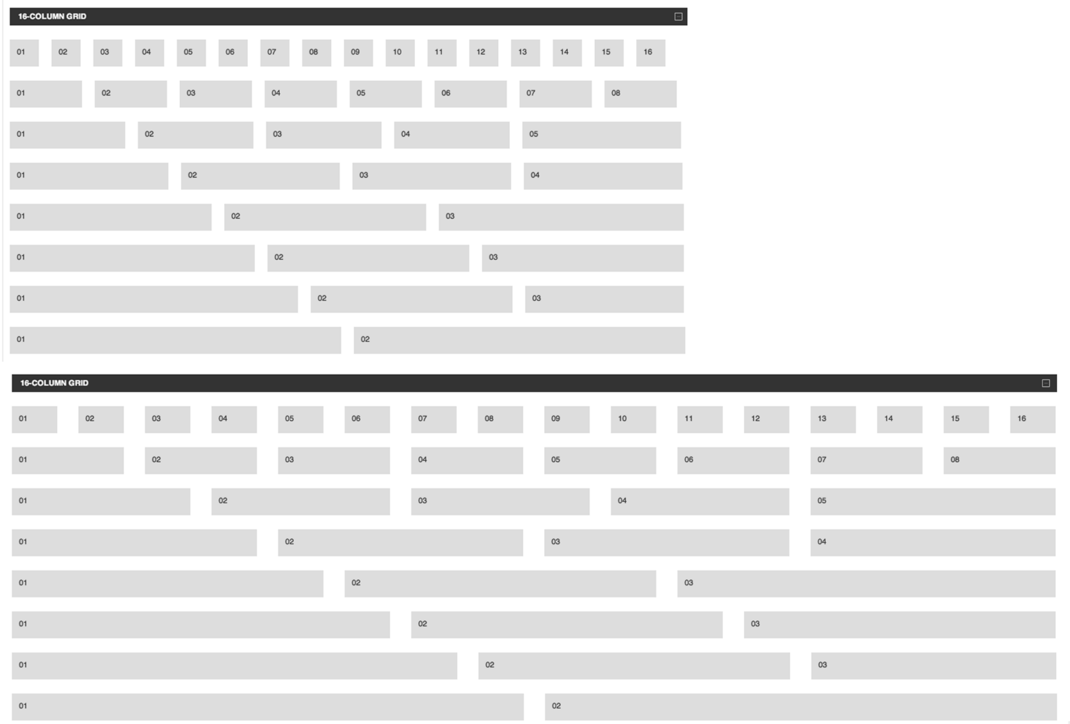


图3-4　960流动网格布局

尽管流动布局听上去非常诱人（毕竟，谁会不喜欢灵活呢），但是动态内容的特性（比如图片、图案）让流动布局变得非常难以实现和维护。本章稍后讨论960网格布局和其他系统时，会看几个流动的网格布局。

到目前为止，我们的讲解已经涵盖了CSS网格框架的基本知识点。下一节我们将深入探讨4个流行的网格布局系统以及如何在Compass中使用它们。

3.3　使用 Blueprint

Blueprint最早是Olav Bjørkøy在2007年开发的，现在由Joshua Clayton和一个贡献者团队进行维护。Blueprint把一些通用的对网格布局、段落和表格进行样式修饰的CSS技术打包到一个框架中，然后可以在各个项目中通用这个框架。你可以完全使用Blueprint，也可以选择Blueprint中你喜欢的模块进行自定义。大多数设计师在初识Blueprint时都是用它来做网格布局的，所以我们从这里开始吧。

如你在3.1节中看到的，CSS网格布局包含容器、列和槽。很快你会看到，列和槽都是虚拟的，这也就意味着你的标签中不会出现任何列和槽。你只需在标签上指明当前内容应该占据多少列宽以及每列之间的槽宽。

3.3.1 使用原生CSS的Blueprint

首先你需要下载并且解压缩Blueprint的CSS以及支持类的文件到你的项目中，并且在你文档的头部加以引用，如代码清单3-1所示。

代码清单3-1 把Blueprint加到页面中

```
<link rel="stylesheet" href="css/blueprint/screen.css">
<link rel="stylesheet" href="css/blueprint/print.css">
<!--[if lt IE 8]>
  <link rel="stylesheet" href="css/blueprint/ie.css">
<![endif]-->
```

3

在这个例子中，你为显示器和打印媒介都添加了样式，同时还为处理所有IE可爱的怪癖添加了条件样式。这是一种厨房水槽（kitchen sink）[①]的做法：把所有的Blueprint的特性都包含进来，包括它的重置、网格布局、排版以及表格支持。你可以选择仅仅引入你想要的Blueprint模块，但是对于我们现在的演示，这个例子刚刚好。在3.3.3节我们会看看如何在Compass中优化Blueprint。

现在你已经把Blueprint加到页面中，可以创建自己的网格布局系统了。先看一个基本的布局，如代码清单3-2所示。

代码清单3-2 一个基本的Blueprint布局

```
<section class="container">                        <-- ❶ 创建网格布局
  <header class="main span-24">                    <-- ❷ 全宽的头
    Header
  </header>
  <section class="content span-20">                <-- ❸ 主内容区域
    Content
  </div>
  <aside id="sidebar" class="span-4 last">         <-- ❹ 侧边栏
    The last column
  </aside>
  <footer class="main span-24">
    Footer
  </footer>
</section>
```

在上边的例子中，你通过全宽的头部和尾部创建了一个简单的、两列的、类博客样式的布局。首先在把整个网格布局包起来的元素上添加了一个`container`类名❶。通过给header和footer元素设置`span-24`的类名来让它们占满网格布局的整个宽度（网格布局在这个例子中一共占24列）❷。由于你希望在主要内容和侧边栏中间有一个垂直的区分，可以分别为它们设置`span-20`❸和`span-4`的类名。注意，你为侧边栏额外添加了一个`last`的类名❹，这个类名会去掉位于这一列的右边的槽，因为这是行内的最后一列。

我们没时间通阅一遍Blueprint的源码，但是在开始一个Compass的例子之前，有必要了解Blueprint CSS的几点内容，如代码清单3-3所示。

① 这是一个谚语，指无所不包，一应俱全。——译者注

代码清单3-3 Blueprint 例子中使用到的CSS

```
.container {width:950px;margin:0 auto;}      ❶ 设置网格布局的宽
.column,
.span-1,
.span-2,
.span-3,
.span-4,
...                                          ❷ 将列向左浮动，添加槽
.span-24 {float:left;margin-right:10px;}
.last {margin-right:0;}
.span-1 {width:30px;}
.span-2 {width:70px;}
.span-3 {width:110px;}           ❸ 侧边栏
.span-4 {width:150px;}
...                              ❹ 主内容
.span-20 {width:790px;}
.span-21 {width:830px;}
.span-22 {width:870px;}
.span-23 {width:910px;}                      ❺ 不含槽的全宽
.span-24 {width:950px;margin-right:0;}
```

在这个从Blueprint网格模块中提取出来的缩减后的列表中，我们看一下网格布局是如何实现的。容器元素有一个950px的宽，并且在页面中居中❶。接下来，通过`span-x`类名定义列宽的列全部向左浮动，同时通过`margin-right`设置一个右侧的10px的槽❷。接下来，侧边栏和主内容的宽度通过`span-4`❸和`span-20`❹设置。最后，全宽的header和footer享有和容器同样的宽值❺。

如果这是你第一次接触CSS网格布局，你可以看到这里的计算并不复杂。Blueprint通过类名控制列宽以40px（30px的列宽加上10px的槽）的基数，1~24的倍率递增。除了基本的`span-x`类，Blueprint还提供了类似`append-x`、`prepend-x`、`pull-x`、和`push-x`等的类名，用来控制网格布局中水平方向列前、列后的填充或位移，另外也有类似的类用来添加边框或者进行其他微调。到目前为止，我们已经探索了Blueprint背后的一些原理和CSS实现，让我们看看如何通过Compass使应用Blueprint网格布局变得更加容易。

3.3.2 使用Compass应用Blueprint

到目前为止，你已经看到了如何通过CSS实现一个简单的布局，接下来让我们看看如何使用Compass应用Blueprint。创建一个新的Compass项目来开始我们的探索，如代码清单3-4所示。

代码清单3-4 创建一个基本的Blueprint项目

```
compass create simple --using blueprint/basic

directory simple/
directory simple/images/
directory simple/sass/
directory simple/sass/partials/
```

```
directory simple/stylesheets/
   create simple/config.rb
   create simple/sass/screen.scss
   create simple/sass/partials/_base.scss
   create simple/sass/print.scss
   create simple/sass/ie.scss
   create simple/images/grid.png
   create simple/stylesheets/ie.css
   create simple/stylesheets/print.css
   create simple/stylesheets/screen.css
...
```

❶ 以下为主要的样式表

❷ 网格布局设置

除了基本的项目结构，还有几个点很重要，需要特别注意。首先，Compass创建了一个主样式文件screen.scss，并引入了Blueprint❶。接下来，Compass创建了一个_base局部文件，其中包含了你网格布局的所有计算❷。在这个例子中，计算规则跟上面静态CSS的例子一样，都是（30+10）×24的设定，所以不用详看了。你可能会奇怪，为什么会使用blueprint/basic。我们很快会看到，Compass有很多种使用Blueprint的方法。为了这个例子，我们先看看基本的模式。

首先，让我们深入看一下生成的screen.scss文件，如代码清单3-5所示。

代码清单3-5 Blueprint基本模式的默认screen.scss

```
// 这里引入一个全局的重置到任何引入这个样式表的地方
@import "blueprint/reset";

// 需要配置blueprint的话，编辑partials/_base.sass文件。
@import "partials/base";

// 引入所有默认的blueprint模块，以便我们能使用这些模块内的混合器。
@import "blueprint";

// 引入非默认的脚手架模块。
@import "blueprint/scaffolding";

// 根据你的配置生成blueprint框架。
@include blueprint;

@include blueprint-scaffolding;
```

❶ 默认的Blueprint 重置模块

❷ 网格布局设置

❸ 让Blueprint的模块可用

❹ 生成网格布局

❺ 表单和其他的Blueprint元件

在生成的screen.scss文件中，添加了Blueprint的reset模块❶，从局部文件中引入了网格设定❷，引入了Compass中强大的Blueprint混合器❸。接下来准备生成你的网格布局❹并添加一些Blueprint额外的特性，比如说与表格修饰相关的❺。真正的魔力发生在你通过@include blueprint生成网格布局时。让我们通过查看Compass源代码的相关行来了解一下这是如何发生的，如代码清单3-6所示。

代码清单3-6 利用强大Compass的Blueprint生成网格布局

```
@mixin blueprint-grid {
  ...
  // Use these classes (or mixins) to set the width of a column.
```

```
@for $n from 1 to $blueprint-grid-columns {        ❶ 生成span-xx类
 .span-#{$n} {
  @extend .column;
   width: span($n); } }
.span-#{$blueprint-grid-columns} {                ❷ 最后一列的类不需要槽
  @extend .column;
  width: span($blueprint-grid-columns);
  margin: 0; }
...
```

在Compass的Blueprint模块中，一系列的混合器强依赖于`blueprint-grid`混合器，它帮你做了网格布局中的大部分脏活累活。这个混合器基于你在_base.scss局部文件中定义的值处理你网格布局中的计算。它通过遍历列数生成你希望的CSS类名❶。如前面章节中的静态CSS例子，对于最后一列的类处理稍微不同，会省略掉列上的槽❷。如你所见，你可以通过修改base局部文件中的数值，得到一个全新的网格系统。我们仅仅把这个混合器拿出来演示Compass如何支持Blueprint的类，并不意味着你得了解它，如果你愿意，你永远都不需要同这类代码打交道。你根本就无需考虑这部分代码是如何实现的，直接应用Blueprint的类即可。但理解背后的工作原理能更好地应用Compass。

到目前为止，你已经看到了Compass如何快速创建基于类名的Blueprint网格布局，接下来让我们看看其他可选方法。

3.3.3　使用Compass应用无需类名的Blueprint

在上个例子中，我们使用了Compass Blueprint的基本模式：

```
compass create simple --using blueprint/basic
```

Compass同时也提供了其他的一些选项。如果你不喜欢Blueprint类名修饰的方式，而更倾向于在其他的选择器中混合进网格样式，可以选用`blueprint/semantic`：

```
compass create simple --using blueprint/semantic
```

如果比较一下两种类型生成的文件，你会发现只多了一个文件，此外在screen.scss底部多了一些额外引入。

```
// 把这些局部文件合并到screen样式文件中。
@import "partials/page";
@import "partials/form";
@import "partials/two_col";
```

使用这种类型，Compass就不会再生成那些`span-xx`类了。你只需要使用`@column`混合器。Compass在`two_col`的局部文件中提供了一个很好的例子，如代码清单3-7所示。

代码清单3-7　使用Compass默认的两列Blueprint布局

```
#container {                                   ❶ 设置网格布局容器
  @include container; }
#header, #footer {                             ❷ 全宽的头部和尾部
  @include column($blueprint-grid-columns); }
#sidebar {
```

```
  // One third of the grid columns, rounding down. With 24 cols,
  // this is 8.
  $sidebar-columns: floor($blueprint-grid-columns / 3);   <- ❸ 侧边栏使用近似1/3的宽
  @include column($sidebar-columns); }
#content {
  // Two thirds of the grid columns, rounding up.
  // With 24 cols, this is 16.
  $content-columns: ceil(2 * $blueprint-grid-columns / 3);
  // true means it's the last column in the row
  @include column($content-columns, true); } }
```

这个清单中的内容虽然很短，但却足以充分演示让网格布局更快的Compass技术（尤其当你重构时）。为了设置整体的网格，你需要一个容器。例子中，你把相应的行为整合进了`#container`选择器❶。你的头尾元素也以相同的方式通过一个混合器设置成了全宽❷。代码中最神奇的部分是，Compass基于侧边栏和主内容分别占1/3、2/3，计算出了侧边栏和主内容应该占据的单元列的个数❸。通过`floor`和`ceil`方法，你可以进行一些基本的舍入以确保恰当的分配。如果你需要再次改变你的_base.scss局部文件中网格布局的列数，这些代码根本不需要修改，依然能正常工作。

到目前为止，我们通过示例驱动的方式学习了Compass中的Blueprint网格布局。接下来探索一些Compass中其他可以应用的著名CSS网格框架。来看一下960网格系统。

3.4 使用 960 网格布局系统

另一个著名的CSS网格框架是Nathan Smith的960网格系统（如图3-5所示）。这个框架的优点在于灵活性。它的960px的宽设定非常适合已经流行很久的1024px宽的屏幕，960px又被分成了2、3、4、5、6、8、10、12、15、16、20、24、30、32、40、48、60、64、80、96、120、160、192、240、320和480等子值。

960网格系统的工作原理跟我们之前探讨的Blueprint CSS框架的大部分原理都是相同的，除了几个关键的点。首先，960网格系统中的槽在单元列的两边都有，这就意味着无论是第一列还是最后一列在它们的外边框上都有一个槽。其次，960网格系统可以指定容器的范围，这样在同一页中就可以有不同的列数和列宽。960网格系统共有12分、16分和24分三种规范，如图3-6所示。



图3-5　960网格系统

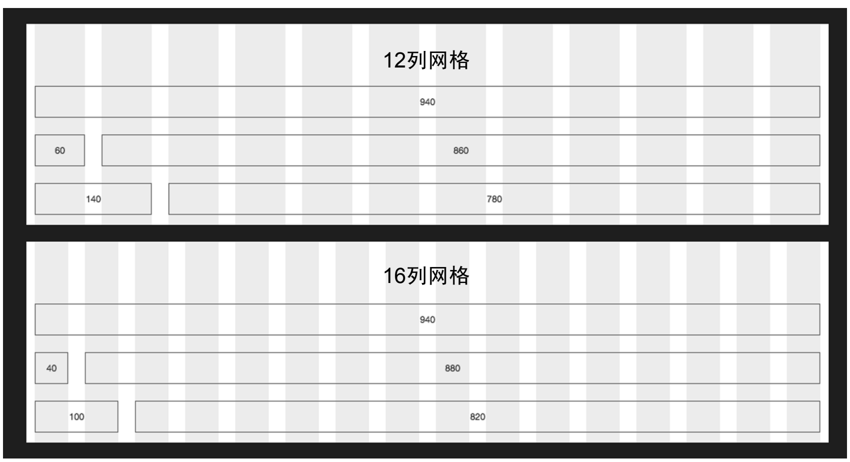


图3-6　960网格系统的例子

知道了这些，让我们重温一下3.3节的例子。

3.4.1 一个基本的 960 布局

像前面章节把Blueprint CSS加入到页面中一样，我们首先要把960的相关文件加入到页面中：

```
<link rel="stylesheet" href="css/reset.css" />
<link rel="stylesheet" href="css/text.css" />
<link rel="stylesheet" href="css/960.css" />
```

默认情况下，你必须把960的重置文件添加到页面中，文本样式文件是可选的，然后引入默认使用12列或者16列布局的网格系统文件，如代码清单3-8所示。

3

代码清单3-8 一个基本的960网格系统的12列布局

```
<section class="container_12">                    <-❶ 创建一个网格布局
  <header class="grid_12">
    Header                                     ❷ 全宽的头
  </header>
  <section class="content grid_10">                <-❸ 主内容区域
    Content
  </div>
  <aside id="sidebar" class="grid_2">              <-❹ 侧边栏
    The last column
  </aside>
  <footer class="grid_12">
    Footer
  </footer>
</section>
```

注意，960网格系统中标签的写法跟Blueprint示例非常相似。只是`container`变成了`container_12` ❶，`span-x`类变成了`grid_x` ❷、❸、❹。机敏的读者会发现侧边栏不需要`last`类来指明是位于一行的最后。这是因为960网格系统中所有列两边都有槽，所以就没必要了。960中有一个`omega`类，它的作用跟Blueprint中`last`类很相似，但是这个只用在你强制网格布局中的内容新起一行的时候。

你可以轻易地把你的布局改成24列。首先你需要引入24列的网格系统文件：

```
<link rel="stylesheet" href="css/reset.css" />
<link rel="stylesheet" href="css/text.css" />
<link rel="stylesheet" href="css/960_24_col.css" />
```

把上边960.css的引用改成960_24_col.css。只要换用了正确的网格系统文件，你可以把你的标签也改成24列版本的了，如代码清单3-9所示。

代码清单3-9 一个基本的960网格系统的24列布局

```
<section class="wrapper container_24">             <-❶ 创建一个网格布局
  <header class="main grid_24">
    Header                                     ❷ 全宽的头
  </header>
  <section class="content grid_20">                <-❸ 主内容区域
    Content
```

```
    </div>
    <aside id="sidebar" class="grid_4">          ◁─❹ 侧边栏
      The last column
    </aside>
    <footer class="main grid_24">
      Footer
    </footer>
</section>
```

如预期的那样，你调整容器❶和列❷、❸、❹以适应比例上的改变。尽管这对于那些以前就用过960网格系统的人来说非常容易，但是在我们介绍使用Compass应用960网格系统之前了解一下这些网格选项是非常重要的。接下来一节将展示在Compass中应用960网格系统的强大威力。

3.4.2　在Compass中使用 960 网格布局

对960网格系统的支持并没有绑定在Compass内，所以首先你需要安装Compass插件。回顾一下第3章，我们可以通过Ruby的Gems安装：

```
gem install compass-960-plugin
```

现在准备创建你的Compass项目吧，如代码清单3-10所示。

代码清单3-10　生成一个新960网格系统的Compass项目

```
compass create -r ninesixty twelve_col --using 960     ◁─ 引入相应插件，
directory twelve_col/                                   ❶ 应用960类型
directory twelve_col/sass/
directory twelve_col/stylesheets/
   create twelve_col/config.rb
   create twelve_col/sass/grid.scss              ◁─❷ 网格布局设定
   create twelve_col/sass/text.scss
   create twelve_col/stylesheets/grid.css
   create twelve_col/stylesheets/text.css
```

当你使用Compass生成一个新的960网格系统项目时，你需要引入相应的插件❶，同时应用相关的类型，以告诉Compass使用何种模板生成。默认情况下，这个插件创建两个样式表，其中之一是伴随960的网格设置❷和基本段落模块。一般的做法是把它们转换成局部文件，在一个screen.scss样式文件中引用它们以减少网络请求负载。为了这个例子的演示，让我们来看一下Compass都为你生成了哪些网格设置，如代码清单3-11所示。

代码清单3-11　Compass生成的960网格系统布局默认的网格设置

```
@import "compass/reset";
@import "960/grid";

// 下边的代码生成由960网格系统提供的默认网格布局
.container_12 {                                ◁─
  @include grid-system(12); }                  ❶ 设定12列的网格类
```

```
.container_16 {
  @include grid-system(16); }

// 但是大多数compass用户倾向于像下边这样构造语义化的布局
// (头尾部占用两倍宽)

$ninesixty-columns: 24;

.two-column {
  @include grid-container;
  #header, #footer {
    @include grid(24); }
  #sidebar {
    @include grid(8); }
  #main-content {
    @include grid(16); } }
```

❷ 设定16列的网格类

❸ 设置混合器中会用到的变量为24列

重新回忆一下3.2.2节中要语义还是要实用的争论。默认情况下，960网格系统的Compass插件支持3种规格的网格布局。它包含了基于类名的12和16列的网格布局❶、❷以及基于混合器的（语义的）24列的网格布局。这就意味着你可以随意选择，要么使用类名，要么使用混合器为你已有的选择器添加网格样式。让我们修改一下这个样式表来适应我们的标签，如代码清单3-12所示。

代码清单3-12 修改960以适应我们的简单网格布局

```
@import "compass/reset";
@import "960/grid";

// 下边的代码生成由960网格系统中css提供的默认网格布局
.container_12 {
  @include grid-system(12); }

// 但是大多数compass用户倾向于像下边这样构造语义化的布局
// (头尾部占用两倍宽)

$ninesixty-columns: 24;

.wrapper {
  @include grid-container;
  header.main, footer.main {
    @include grid(24); }
  #sidebar {
    @include grid(4); }
  .content {
    @include grid(20); } }
```

❶ 移除16列网格，你不需要它

❷ 设置24列网格

为了符合我们的要求，把16列的版本移除掉，因为我们并不需要❶。由于12列格子不满足我们的需求，所以我们把格子设置成24列❷。假如你倾向于使用类名来调用24列的版本，你可以添加一个对grid-system混合器的调用：

```
.container_24 {
  @include grid-system(24); }
```

简单却强有力。若你想了解更多关于Compass 960网格系统插件的用法和特性，可以访问项目的源代码，位于GitHub上：https://github.com/chriseppstein/compass-960-plugin。

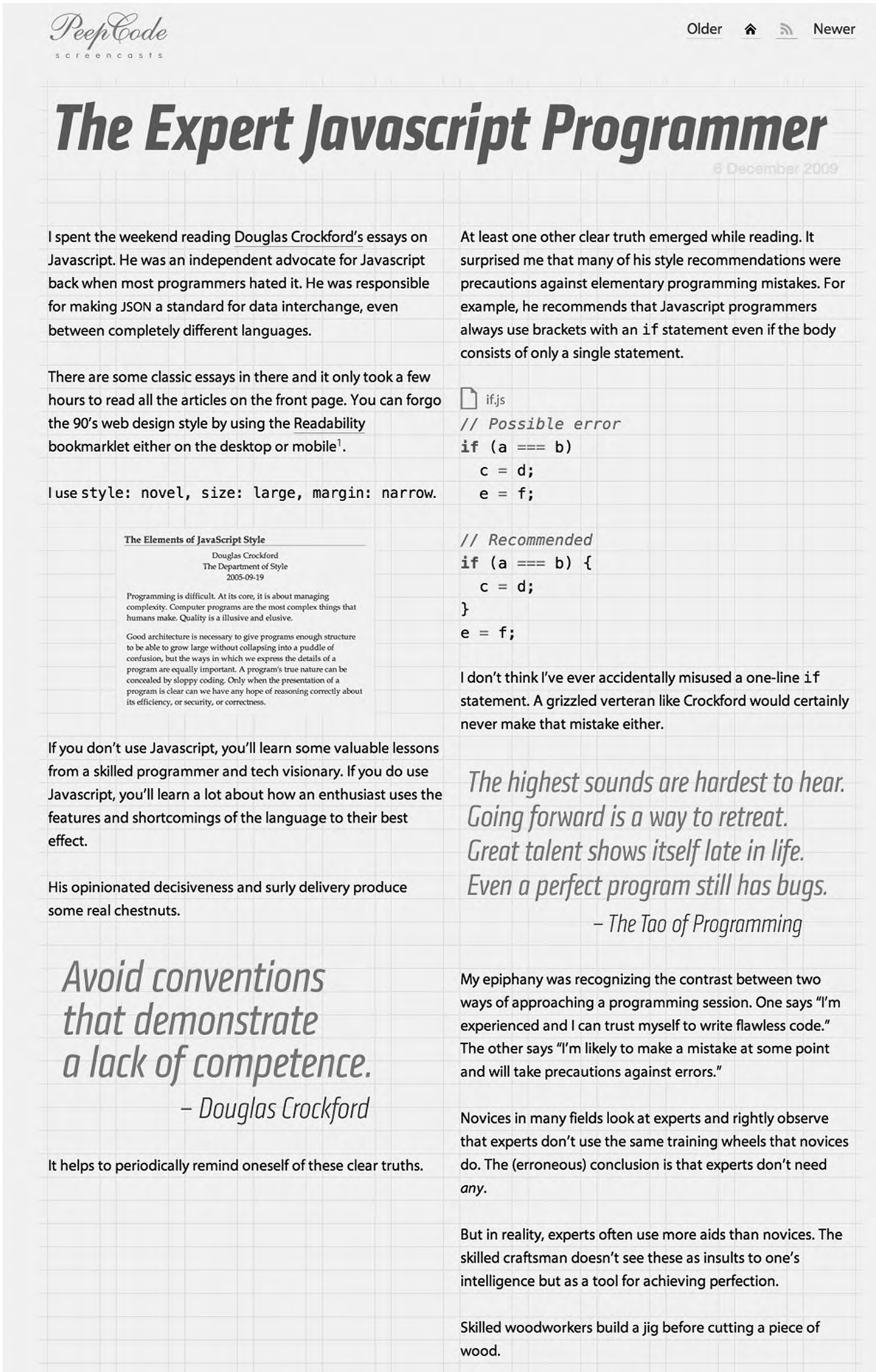
到目前为止，我们关于网格布局框架的讨论都集中在网格中内容的垂直排布上。你已经知道了如何使用容器设置一个网格布局。我们也看到了如何通过CSS类名或者Sass混合器来轻松地在网格布局中排布内容；探索了如何下载Compass的插件以提供更多的网格布局支持。下一节将看看一个经常被伟大网格设计所忽视的问题——段落垂直韵律。

3.5　通过 Compass 处理垂直韵律

在前一节，我们看到了CSS网格布局如何帮助管理垂直列之间内容的空隙。许多设计师只关注这里，往往忽略了页面中其他地方的留白，比如说行与行之间内容的留白。我们提到的内容往往指文本，但是也包括图片、视频、表格以及你设计中的任何其他元素。就像网格布局通过统一的槽来很好地排布垂直列之间的内容一样，一个好的网格布局也应该管理*垂直韵律*，采用均匀的水平留白。但垂直韵律看起来到底是怎么样的呢？我们回到这一章开头的PeepCode博客，如图3-7所示。

注意，尽管标题内容、代码列表以及引述的大小不同，但是它们都在水平格子线上排布得很好。如果你把页面与一张乐谱相对比，会发现网格布局的线很像节拍线。主体文本内容沿着节拍线形成一个韵律。标题内容、图片、表格以及其他的块元素既可以位于节拍线的上面，又可以位于下面，甚至是后面，但是主体文本内容不能偏离页面韵律。那么，如何实现这种效果呢？

首先你需要设置行高或者说主体文本之间的*行间距*。行间距是你的文本连续基线之间的距离或者说是你的垂直韵律，再或者说是你在页面中添加的水平留白的单元。这就意味着所有元素的高（即字体大小、行高、上下内间距以及上下外间距的值综合起来）应该是这个基本单元的倍数。接下来几节将构建一个布局并在其中应用垂直韵律。我们演示每一步中需要用到的CSS，以及在更短时间内实现相同效果的Compass的快捷做法。



PeepCode screencasts

Older Newer

The Expert Javascript Programmer

6 December 2009

I spent the weekend reading Douglas Crockford's essays on Javascript. He was an independent advocate for Javascript back when most programmers hated it. He was responsible for making JSON a standard for data interchange, even between completely different languages.

There are some classic essays in there and it only took a few hours to read all the articles on the front page. You can forgo the 90's web design style by using the Readability bookmarklet either on the desktop or mobile[1].

I use `style: novel, size: large, margin: narrow`.

The Elements of JavaScript Style

Douglas Crockford
The Department of Style
2005-09-19

Programming is difficult. At its core, it is about managing complexity. Computer programs are the most complex things that humans make. Quality is a illusive and elusive.

Good architecture is necessary to give programs enough structure to be able to grow large without collapsing into a puddle of confusion, but the ways in which we express the details of a program are equally important. A program's true nature can be concealed by sloppy coding. Only when the presentation of a program is clear can we have any hope of reasoning correctly about its efficiency, or security, or correctness.

If you don't use Javascript, you'll learn some valuable lessons from a skilled programmer and tech visionary. If you do use Javascript, you'll learn a lot about how an enthusiast uses the features and shortcomings of the language to their best effect.

His opinionated decisiveness and surly delivery produce some real chestnuts.

> Avoid conventions that demonstrate a lack of competence.
>
> – Douglas Crockford

It helps to periodically remind oneself of these clear truths.

At least one other clear truth emerged while reading. It surprised me that many of his style recommendations were precautions against elementary programming mistakes. For example, he recommends that Javascript programmers always use brackets with an `if` statement even if the body consists of only a single statement.

if.js

```
// Possible error
if (a === b)
  c = d;
  e = f;

// Recommended
if (a === b) {
  c = d;
}
e = f;
```

I don't think I've ever accidentally misused a one-line `if` statement. A grizzled verteran like Crockford would certainly never make that mistake either.

> The highest sounds are hardest to hear.
> Going forward is a way to retreat.
> Great talent shows itself late in life.
> Even a perfect program still has bugs.
>
> – The Tao of Programming

My epiphany was recognizing the contrast between two ways of approaching a programming session. One says "I'm experienced and I can trust myself to write flawless code." The other says "I'm likely to make a mistake at some point and will take precautions against errors."

Novices in many fields look at experts and rightly observe that experts don't use the same training wheels that novices do. The (erroneous) conclusion is that experts don't need *any*.

But in reality, experts often use more aids than novices. The skilled craftsman doesn't see these as insults to one's intelligence but as a tool for achieving perfection.

Skilled woodworkers build a jig before cutting a piece of wood.

图3-7 PeepCode博客的垂直韵律

3.5.1　确定基线

像上一节中提到的那样，我们首先统一一下，为主体文本选择一个易读的基本字号和默认的行高。

```
body {
  font-family: 'Helvetica Neue', sans-serif;
  font-size: 16px;
  line-height: 24px;
}
```

通过一个简单的CSS重置，这一小段CSS初始化了一个看上去很不错的1.5em的基线，如图3-8所示。

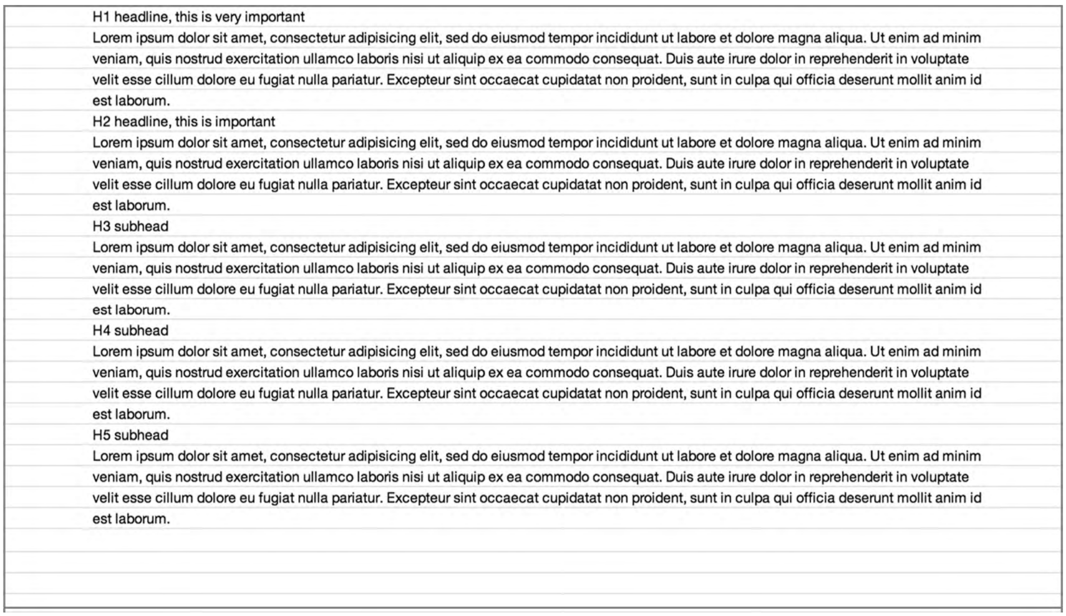


图3-8　简单的垂直韵律示例

为了让你诚实，我们在页面背景上使用一个重复的图片，和基线一样高，也是24px。现在让我们继续开发你的设计。你需要在标题内容和主体内容上添加大小对比。让我们为`<h1>`到`<h5>`的标题内容确定段落比例的大小，如代码清单3-13所示。

代码清单3-13　设置行高

```
h1 {font-size: 48px;}
h2 {font-size: 36px;}
h3 {font-size: 24px;}
h4 {font-size: 20px;}
h5 {font-size: 18px;}
h1,h2,h3,h4,h5 {line-height: 1.5em;}          <- ❶
p {margin: 1.5em 0;}
```

如图3-9所示，你已经为标题内容设定一个合适的高度❶，并且在标题和段落之间添加了留白。



图3-9　设置段落的比例

如你在图中所见，对标题内容已经有了一个很好的排布，但是你的主体文本并没有与你预期的垂直韵律同步。为了修复这个问题，需要让标题内容和其他元素的高是你的基线的整数倍。在本例中，这意味着标题内容的字体大小、行高、垂直外间距以及垂直内间距全部综合起来应该是你的基线单元24px的整数倍。调整一下你的标题内容样式然后回到韵律上来。调整行高以适应不同的字体大小使其始终保持在基线上的公式是：

```
(baseline unit/ font-size) = line height
```

因为你的<h1>元素的设定已经是基线的整数倍了，我们拿例子中的<h2>来演示。计算应该如下：

```
    (24px / 36px) = .6666667 em

h1 {font-size: 48px; line-height: 1.5em}
h2 {font-size: 36px; line-height: .666667em}
h3 {font-size: 24px; line-height: 1em}
h4 {font-size: 20px; line-height: 1.2em}
h5 {font-size: 18px; line-height: 1.33333em}
p {margin: 1.5em 0}
```

调整每一个标题样式的行高以便重回垂直韵律，如图3-10所示。

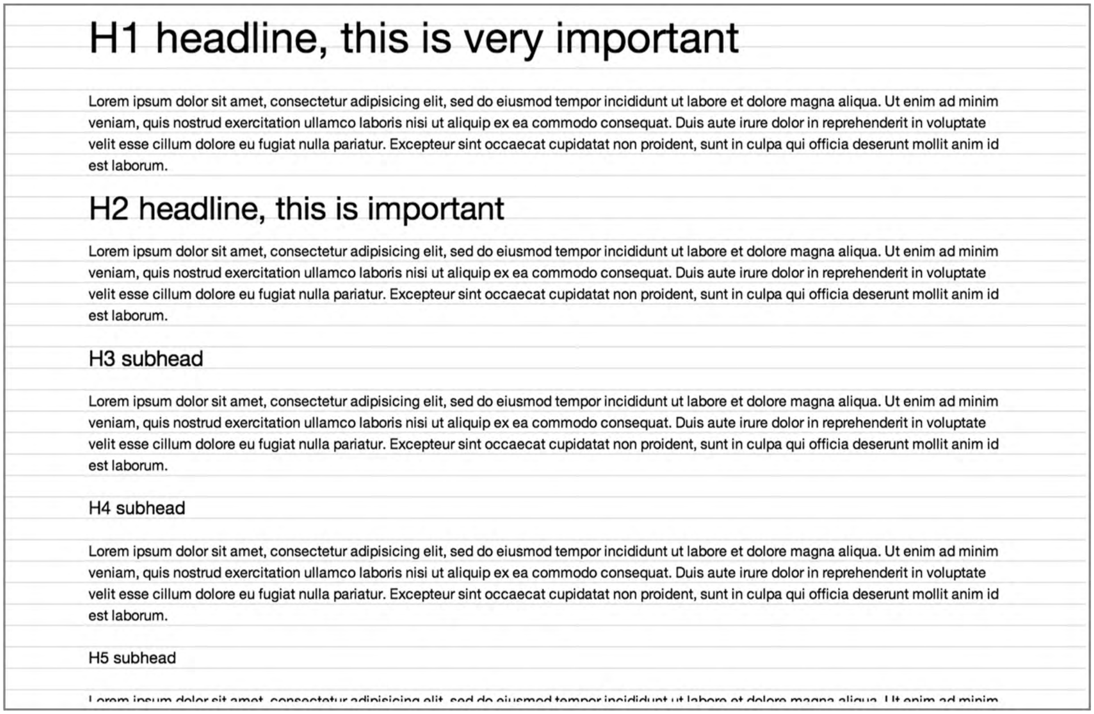


图3-10　垂直韵律的版式规模

如你所见，主体文本也重新回到了基线上。现在这对你来说计算应该不难，但是计算始终得算。为什么不让电脑帮你做这部分繁重的工作呢？它可以把你从计算每一个元素的行高中解放出来，同时也允许你试验不同的字体大小，让样式表根据你的修改而调整。现在看看如何用Compass实现这样的样式，如代码清单3-14所示。

代码清单3-14　使用Compass处理垂直韵律

```
@import "compass/typography";

$base-font-size: 16px;
$base-line-height: 24px;
@include establish-baseline;

body {
  font-family: 'Helvetica Neue', sans-serif;
  @include debug-vertical-alignment("../images/debug.png");
}

h1 {@include adjust-font-size-to(48px)}
h2 {@include adjust-font-size-to(36px)}
h3 {@include adjust-font-size-to(24px)}
h4 {@include adjust-font-size-to(20px)}
h5 {@include adjust-font-size-to(18px)}
p {margin: 1.5em 0;}
```

❶ 引入段落模块
❷ 声明字体大小
❸ 引入调试混合器

Compass的实现需要首先引入Compass的段落模块❶。接下来声明基本字体和基本行高两个变量❷。Compass通过一个混合器为页面添加网格调试图片❸。最后通过调用`adjust-font-size-to`混合器来设置字体和行高。Compass使用跟本节前面提到的计算方法相同的公式。这个例子真正的优势在于灵活性。现在你可以试验任意的基本字体大小、基本行高或者元素字体大小，正确的结果会在编译时计算出来。现在你已经看了一个构建基线的简单示例，我们再看看当你需要时，Compass提供的可以添加额外留白的附加辅助器。

3.5.2 前置和后置留白

Compass的`establish-baseline`和`adjust-font-size-to`混合器很容易让你的文本位于一个垂直韵律上。但是当你需要添加额外留白时，如何在不重新计算的前提下保持韵律呢？幸好，Compass也为这样的场景提供了辅助器。让我们看看之前CSS例子中的段落样式：

```
p {margin: 1.5em 0;}
```

你通过上下外间距使段落相互分开一些，让浏览页面变得简单。如果你想对一个很重要的标题内容做同样的事呢？

```
h2.important {margin: 1.5em}
```

问题在于，除非这个元素的字号恰好是你的基线韵律的整数倍，否则当你的主体文本偏离韵律时，你会在设计中遇到一些不和谐音符。Compass提供了一些辅助器来给你的元素添加前置和后置留白以维持你的韵律。你可以按照这种方法处理先前的任何一个元素：

```
p {@include leader; @include trailer;}
h2.important {@include leader(2); @include trailer(2)}
```

`leader`混合器在元素前加一个基线单位的外间距，类似地，`trailer`混合器在元素的后边加一个基线单位的外间距。需要更多？你可以给混合器传递你期望的基线单元个数作为参数。如果你希望用内间距代替外间距，Compass也提供了`padding-leader`和`padding-trailer`这两个变种混合器，它们使用内间距而不是外间距添加留白。

3.6 小结

在这一章，我们看了流行的CSS网格框架如何简化维护留白和快速构建原型设计。通过简单添加一些CSS类，你就能创建彼此之间有统一间距的竖列内容。你也看到了相比单纯使用静态CSS，Compass如何进一步简化使用和创建网格框架。最后，你学习了如何使用Compass的段落模块中的韵律辅助器来管理页面中的行间距等留白。下一章将深入探索其他Compass辅助器，利用它们能快速完成CSS常见的枯燥乏味的工作。

第 4 章

有Compass就不再枯燥

本章内容

- ❑ 使用Compass重置浏览器默认样式
- ❑ 改进样式表排版的Compass辅助器
- ❑ 使用Compass创建粘滞的页脚、多样化的表格以及浮动

我们已经对Sass语法有了初步了解，也知道Compass什么时候能派上用场，现在让我们百尺竿头更进一步。这一章将介绍一些你每天都会遇到、很简单、很常见（也很无聊）的任务，看看怎么通过Compass节省自己的时间和精力，同时汲取社区智慧。如果你还在手写CSS，没有把思维方式转换为编写动态样式表，那么你一定体会得到，有些样式表工作做起来像被千刀万剐了一样难受。比如重置CSS样式，把列表元素改成水平导航，设置链接的颜色，把主标题文字换成一张图片等，这些任务在每个项目里都被重复了无数次。而本章会展示一些Compass提供的辅助器，让这些工作更快、更方便、更适用。

4.1 一张更好的白纸源自有针对性的样式重置

在第1章里，我们看过了CSS样式重置以及如何在Compass项目里仅仅靠添加`@import "compass/reset"`即可引入Eric Meyer的样式重置v2.0。即便这很方便，有时全局样式重置的量级还是显得略重。幸好，Compass提供了更多粒度的方案。选择的关键在于你需要全局样式重置，还是只需要有针对性的重置。

4.1.1 全局样式重置

如果你之前已经使用过CSS样式重置，那可能是一个全局的样式重置，它利用刀耕火种似的手段去除浏览器固有的HTML元素样式。它的流行是因为可以给Web应用程序造出一张绝对一致的白纸。当你构建一个传统的Web应用程序时，如果你需要支持很多浏览器，甚至包括低版本的Internet Explorer，那么一个全局的样式重置是非常给力的。Compass基于Eric Meyer的经典方案，提供了一个全局的样式重置，并恰如其分地命名为`global-reset`。而当使用`global-reset`之前，你有必要理解其内部的混合器，它们其实跟Compass的任何混合器都是一样的，如代码清

单4-1所示。

代码清单4-1　Compass中的global-reset

```
@mixin global-reset {
  html, body, div, span, applet, object, iframe,
  h1, h2, h3, h4, h5, h6, p, blockquote, pre,
  a, abbr, acronym, address, big, cite, code,
  del, dfn, em, img, ins, kbd, q, s, samp,
  small, strike, strong, sub, sup, tt, var,
  b, u, i, center,
  dl, dt, dd, ol, ul, li,
  fieldset, form, label, legend,
  table, caption, tbody, tfoot, thead, tr, th, td,
  article, aside, canvas, details, embed,
  figure, figcaption, footer, header, hgroup,
  menu, nav, output, ruby, section, summary,
  time, mark, audio, video {

    @include reset-box-model;
    @include reset-font; }
  body {
    @include reset-body; }
  ol, ul {
    @include reset-list-style; }
  table {
    @include reset-table; }
  caption, th, td {
    @include reset-table-cell; }
  q, blockquote {
    @include reset-quotation; }
  a img {
    @include reset-image-anchor-border; }

  @include reset-html5; }
```

定义全局样式重置

包含个别的混合器

注意，global-reset只是一个在内部应用了若干样式重置混合器的包装。这些（通过Sass的@include引入的）混合器不仅致力于解决浏览器在盒模型、排版、列表样式、表格样式等诸多方面的不一致问题，还为新的HTML5元素添加了默认样式。我们会在下一节看到这些特殊的混合器，但在结束对global-reset混合器的讨论之前，有必要提一下，Compass的样式重置仅需要导入compass/reset，这意味着什么呢？为了搞清楚这一点，来看一看compass/reset的源代码：

```
@import "reset/utilities";

@include global-reset;
```

原来如此，只有两行。第一行的@import使global-reset混合器里面的混合器都可用。第二行@include引入了这个全局样式重置。下一节，我们来看一看在没有这个全局样式重置的情况下，如何实现有针对性的样式重置。

4.1.2　通过有针对性的样式重置进行更多控制

假设你有一个移动端的界面，那么全局的样式重置其实会给它带来不必要的负担。可能你的页面并没有用到`<table>`、`<blockquote>`或列表。因为页面的负担是很关键的，样式表瘦身带给移动应用用户体验的影响比桌面端更加明显。如果你用普通的方式写CSS，你可能会收藏一段全局样式重置的代码片段，然后把它用在一个个项目之中。如果你只想在新项目中用一部分，那就需要把它摘到一个文本编辑器里，再去掉你不需要的部分。比这个更进一步的方式是使用`@import "compass/reset/utilities"`，而不必使用全局样式重置。让我们看几个上一节中出现的组成全局样式重置的混合器。

1. 高瞻远瞩的HTML5样式重置

HTML5的一个令人兴奋的新特性，就是它引入了新的标记。你可以在适当的地方把一些`<div>`换成`<header>`、`<footer>`以及`<nav>`等更语义化的标记。可惜，在如何对待这些新标记的问题上，浏览器之间没有完全达成一致。设想一下，你需要列出每个新标记并逐个应用`display: block`。你可以把这个列表背下来，甚至可以存一段代码然后不断地复用。而如果你使用Compass，那只使用`reset-html5`混合器就可以快速完成这项工作。我们来看一看Compass的`reset-html5`混合器，如代码清单4-2所示。

代码清单4-2　HTML5样式重置

```
@mixin reset-html5 {
  article, aside, details, figcaption, figure,
  footer, header, hgroup, menu, nav, section, summary {
    display: block; } }
```

现在，在你的SCSS文件里写入`@include reset-html5`调用这个混合器，你就不必记住这11个标记了。

2. Compass文档中更多的样式重置

样式表中的全局样式重置和HTML5样式重置基本上可以覆盖90%的用例了，但我们还是鼓励你去查阅Compass文档中样式重置的完整列表。表4-1给出了一个快速的预览。

表4-1　Compass里可用的样式重置

样式重置混合器	目　　的
`reset-box-model`	移除元素的内外边距和边框
`reset-font`	重置文字的字号和基线
`reset-focus`	移除浏览器提供的轮廓线（比如Safari给`<input>`元素加上的那一圈线）
`reset-table` 和 `reset-table-cell`	重置表格的边框和对齐方式
`reset-quotation`	为`<blockquotes>`添加仅存在于样式表中的双引号

现在，你已经知道如何移除浏览器的样式了。我们看一看如何在一般场景下使用Compass的辅助器添加你自己的样式。

4.2 更快更直观的排版工具

可能没有任何东西比排版更能影响你的设计了。排版不仅仅是选择字体和字号。对于平面设计师来说，设置列表样式、处理文本折行都要花费大量时间。因为Web是一个交互的、数据驱动的媒介，Web设计师有了更多的选择，比如设置优雅的超链接样式和被截断文字的样式。我们在样式表中处理排版，一部分工作是选择产出什么样的设计，另一部分工作是将其实现。Compass在此帮助大家完成了后半部分的工作，让你快速完成设置链接、列表以及其他元素样式的枯燥工作，同时专注在设计本身。在下一节，我们会看到一些帮助你设置超链接样式的混合器，也是首先要介绍的排版辅助器。这部分示例都需要你通过`@import "compass/typography"`使用Compass的排版模块。

4.2.1 起锚远航：链接辅助工具

好的设计善于利用对比和反差。给超链接一个特殊的样式，令其在网页中脱颖而出不仅仅是一个审美问题，更是一个用户体验问题。因此，优秀的设计师（在样式重置之后）在创建一个新样式表时，往往会定义基础的文本和链接的颜色——Compass可以把这个轻松的工作变得更轻松。

1. 轻轻松松为链接配色

许多被CSS社区认可的模式都是经过千锤百炼的，它们旨在创造可靠的跨浏览器的设计。其中一个模式就是建议的根据链接的不同状态（如`:hover`和`:visited`）在样式表中排列超链接伪类的顺序。链接的不同状态是浏览器为链接的伪类赋予优先级的一种有趣方式。比如，如果你把鼠标停放在一个已经访问过的链接上，哪个样式会生效？

最佳实践是按照如下顺序先后包含伪类选择器：

1. `<a>`
2. `:visited`
3. `:focus`
4. `:hover`
5. `:active`

这意味着你的CSS看起来应该像下面这样，如代码清单4-3所示。

代码清单4-3 根据浏览器特征设置链接颜色的CSS

```
a {color: #333}
a:visited {color: #555}
a:focus {color: #f00}
a:hover {color: #00f}
a:active {color: #f00}
```

你当然可以死记硬背地记住这个顺序，然后写伪类来改变链接颜色。Compass提供了一个方便的混合器来处理这项工作：

```
a { @include link-colors(#333, #00f, #f00, #555, #f00); }
```

细心的读者会注意到Compass例子中的这个颜色顺序和CSS例子中的并不吻合。因为Compass选择了针对生产效率优化的顺序，而不是按浏览器的优先级顺序。这个`link-colors`的颜色参数顺序是你最可能在样式表里使用的顺序。表4-2展示了`link-colors`完整的参数顺序和它们被应用到不同链接状态的顺序。

表4-2　**`link-colors`**的参数

`link-colors`顺序	浏览器顺序
`<a>`	`<a>`
`:hover`	`:visited`
`:active`	`:focus`
`:visited`	`:hover`
`:focus`	`:active`

如果你是一个注重细节，喜爱简洁，更喜爱可读性的人，你可能会这样按名称传递参数：

```
a { @include link-colors(
          #333,
          $hover: #00f,
          $active: #f00,
          $visited: #555,
          $focus: #f00);
}
```

不命名的语法有其优越性。因为`link-colors`参数是可选的，遵循帕累托（Pareto）著名的"二八原则"（在这里意味着20%的代码覆盖80%的用例），你可以只传递前两个参数来定义默认的颜色和`:hover`状态的颜色：

```
a { @include link-colors(#333, #00f); }
```

现在，你已经看到如何给链接着色了。让我们看一看在设计时，Compass让你从视觉上便于区分链接的其他方式。

2. 通过**`hover-link`**设置悬停样式

有些可用性专家建议链接下面始终要有下划线，以此提示用户这个东西可以点击。但在一些行高有限的情况下，增加下划线反而会降低阅读体验。假设你希望用下面的CSS实现仅当用户悬停在链接上时才显示下划线：

```
a { text-decoration: none}
a:hover { text-decoration: underline }
```

Compass让链接只在`:hover`时加下划线变得更简单了，仅仅使用`hover-link`混合器即可：

```
a {@include hover-link}
```

看一眼这个混合器，就知道它会做什么，简单吧？不必看样式表里用了什么属性，只要看它的名字就行了：`hover-link`。

3. 通过**`unstyled-link`**设置隐性的链接

现在设想你希望在一段话里隐藏一个链接，移除所有会提示用户这是一个链接的样式。你可以这样写CSS：

```
p.secret a {
  color: inherit;
  cursor: inherit;
  text-decoration: inherit
}
```

这段代码会去掉所有颜色、光标样式或下划线，把你的文本混入文本容器中。但是`:hover`和`:focus`状态下它会怎么样呢？让我们更新一下CSS：

```
p.secret a,
p.secret a:hover,
p.secret a:focus {
  color: inherit;
  cursor: inherit;
  text-decoration: inherit
}
```

这个建议不错，但是Compass通过`unstyled-link`混合器把这个工作变得更简单了：

```
p.secret a { @include unstyled-link }
```

同样，Compass混合器的名称描述了它的用途。现在你已经看到Compass是如何让链接的样式工作更简单了。让我们再看看它如何让列表的样式工作变得更简单。

4

4.2.2 创建各种各样的列表

在Web排版中，处理列表元素是一个无法回避的问题。在这个媒介中，简洁明了成为了沟通的关键，作为一个设计师，`<ul>`是你的好朋友。在本节，我们会看到一些Compass的辅助器（Sass混合器），在设计优秀的列表时，它们可以快速处理各种常见的工作。

1. 用`pretty-bullets`装点列表

基于图片的项目符号（如图4-1所示）可以为你的列表元素增加冲击力。但是IE从5.5开始支持的`list-style-image`属性有很多问题。比如在低于IE8的版本中，浮动列表元素不会显示列表项的图片。为了找到跨浏览器的解决方案，设计师们经常使用背景图片来显示列表的项目符号：

```
ul.features li {
  background: url(/images/pretty-bullet.png) 5px 5px no-repeat;
  list-style-type: none;
  padding-left: 20px;
}
```



图4-1 美观的项目符号示例

一段小小的代码足以让你头疼。首先，你不得不根据预期的内边距和图片宽度计算布局。这

里的5px 5px是background快捷方式中background-position部分的x和y坐标值，它是按照下面的公式计算出来的：

```
# x = (padding - image width) / 2
# y = (line height - image height) / 2
```

第二个问题是由第一个问题引申出来的：你必须知道你的图片尺寸。考虑到这些问题，Compass提供了pretty-bullets混合器，有了它，通过背景图片显示列表的项目符号就变得容易了：

```
ul.features {
  @include pretty-bullets('pretty-bullet.png')
}
```

通过pretty-bullets混合器，Compass解决了最繁重的工作，从图片那里得到其尺寸，进行运算，得出和之前例子相同的CSS。如果你需要做更多的控制，可以根据名称或次序传递给定的$height、$width和$padding参数：

```
ul.features {
  @include pretty-bullets('pretty-bullet.png',
            $padding: 10px,
            $line-height: 22px)
}
```

注意在每个Compass示例中，我们并没有定义完整的图片路径，只是写了文件名和扩展名。这是因为pretty-bullets混合器使用了image-url辅助器,它可以在开发环境和生产环境返回各自不同的完整路径。现在你已经看到了如何创建美观的、跨浏览器的项目符号，让我们再看看如何去掉列表的项目符号。

2. 通过no-bullet和no-bullets去掉项目符号

Compass同样提供了一些快速移除<li>元素列表样式的方式。你可能会想，为什么不使用list-style: none呢?对于IE8以上版本（以及其他浏览器）可以这样做。如果你需要支持雷蒙德制造的8以下版本的浏览器[①]，那么你需要使用如下方式：

```
li.no-bullet {
  list-style-image: none;
  list-style-type: none;
  margin: 0px;
}
```

有了Compass，你不必记住这个写法，使用no-bullet混合器即可：

```
li.no-bullet { @include no-bullet }
```

如果你希望去掉整个列表的项目符号，也可以使用复数形式的no-bullets混合器：

```
ul.no-bullet { @include no-bullets}
```

这个写法会为列表中每个<li>单独混入no-bullet混合器。现在你已经看到如何为列表定制项目符号或去掉项目符号了，让我们看一看如何把列表“放倒”。

① Redmond，微软总部，这里暗指IE。——译者注

3. 轻松横向排布

浏览器默认的列表样式是竖直排布列表项，并且有内外边距。这本是很好的显示方式，设计师们还常常喜欢横向排布包含导航链接的列表（如图4-2所示）。



图4-2 横向列表示例

考虑下面这段标记：

```
<ul class="nav">
  <li><a href="/">Home</a></li>
  <li><a href="/services">Services</a></li>
  <li><a href="/blog">Blog</a></li>
  <li><a href="/contact">Contact</a></li>
</ul>
```


如何通过CSS把这个列表转成横向排布的导航栏，并且保证列表项之间有8像素的间距呢？我们通常建议像代码清单4-4一样写CSS。

代码清单4-4 创建<ul>导航的CSS

```
ul.nav {
  border: 0;
  margin: 0;
  overflow: hidden;
  padding: 0;
}
ul.nav li {
  display: inline;
  float: left; /* 横向排布菜单 */
  margin-left: 0px;
  padding-left: 4px;
  padding-right: 4px;
}
```

和本章中的很多工作一样，这项工作并不是很难，但是做多了会厌倦。有了Compass，仅仅通过引入`horizontal-list`混合器就可以完成这一工作。

```
ul.nav { @include horizontal-list }
```

除此之外，Compass还提供了一些钩子，用于设置其第一个元素和最后一个元素的特殊样式。下面是完整的CSS输出，如代码清单4-5所示。

代码清单4-5 Compass `horizontal-list`辅助器输出的CSS

```
ul.nav {
  margin: 0;
  padding: 0;
```

```
  border: 0;
  overflow: hidden;
  *zoom: 1;
}
ul.nav li {
  list-style-image: none;
  list-style-type: none;
  margin-left: 0px;
  white-space: nowrap;
  display: inline;
  float: left;
  padding-left: 4px;
  padding-right: 4px;
}
ul.nav li:first-child, ul.nav li.first {
  padding-left: 0;
}
ul.nav li:last-child {
  padding-right: 0;
}
ul.nav li.last {
  padding-right: 0;
}
```

`.first`是针对低版本浏览器的

`.last`是针对低版本浏览器的

对于支持`:first-child`和`:last-child`的浏览器，我们略去了两边元素对外的内边距。而对低版本浏览器，我们可以使用`.first`和`.last`这两个类名。

直到现在，除了知道会少写很多代码，你可能还没有发现这个混合器的关键所在。Compass中的大多数混合器都一样，关键在于都利用了Sass混合器的动态特性。`horizontal-list`混合器有两个参数：`$padding`和`$direction`。因为它们都是可选的，所以你可以忽略它们构建一个从左到右、间距为8像素的列表。（实际上`$padding`的默认值是`4px`。左边4像素加右边4像素就是8像素。）如果你想颠倒项目顺序并加宽边距该怎么办呢？很简单，给这个混合器传入两个参数：

```
ul.nav { @include horizontal-list(7px, right)}
```

现在，列表项两边就都有了7像素的边距（两个项目间隔14像素）并且右浮动，顺序颠倒。下面是更新之后的CSS代码片段：

```
...
ul.nav li {
  ...
  float: right;
  padding-left: 7px;
  padding-right: 7px;
}
```

如你所见，Compass把横向导航列表减少到了一行代码，那么Compass可以处理内联列表吗？

4. 用`inline-list`处理内联列表

前面有一节，我们介绍了如何让链接看起来像文字。列表也可以这样吗？看看下面这段代码：

```
<ul class="giant-words">
  <li>Fee</li>
  <li>Fi</li>
  <li>Fo</li>
  <li>Fum</li>
</ul>
<p>are some words of giants with acute senses of smell
for Englishmen.</p>
```

在这个（人为设计的）例子中，我们把列表设置成内联的样式，并用逗号隔开，这样可能读起来更合适。Compass可以用一行代码就做到：

```
ul.words { @include delimited-list }
```

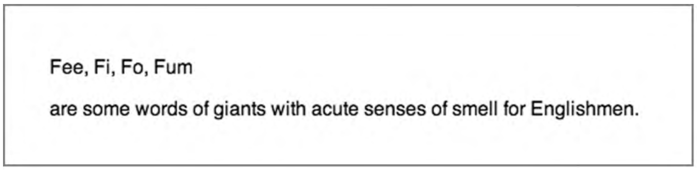


图4-3　分隔列表示例

通过组合`:after`和`:last-child`（最后一个列表项是没有逗号的），Compass产生的CSS可以让列表变成内联的。

但稍等，这还不够！因为Compass还可以让你定制分隔符，让这个列表更霸气：

```
ul.words { @include delimited-list("! ") }
```

至此，Compass支持的列表样式就讲解完了。下一节，我们看看Compass如何帮助我们做各种文字处理。

4.2.3　用辅助工具征服文字

不同于平面设计师，Web设计师花费大量时间调整他们自己并不写甚至可能永远也看不到的东西。设计模板时要考虑用户提供的、数据驱动的内容，这些内容可能会溢出预设的容器范围。幸运的是，Compass包含了一些辅助器可以让这类工作更简单。

1. 用省略号代表截断内容

Web设计师常常面对的一个问题是把一段长度不确定的文本放入一个宽度固定的容器里，比如一个表格的单元格。在过去，设计师需要在渲染标记之前从服务器端截断内容。有了CSS，你可以应用`text-overflow: ellipsis`：

```
td.dot-dot-dot {
  white-space: nowrap;
  overflow: hidden;
  text-overflow: ellipsis;
}
```

现在，正常情况下会折断或溢出容器的文本会被截断并以省略号取而代之。猜对了，Compass

把这件事变得更简单了：

```
td.dot-dot-dot {
  @include ellipsis;
}
```

使用ellipsis混合器额外的好处是，这样可以生成带Opera和微软等不同浏览器厂商前缀的属性名。完整的CSS输出如下：

```
td.dot-dot-dot {
  white-space: nowrap;
  overflow: hidden;
  -o-text-overflow: ellipsis;
  -ms-text-overflow: ellipsis;
  text-overflow: ellipsis;
}
```

值得注意的是，为了让text-overflow生效，必须同时使用white-space: nowrap。通常情况下，开发者会觉得很难记住它是white-space还是whitespace。谢天谢地，Compass想你之所想，来看看下一节吧。

2. 用nowrap防止文本折行

nowrap混合器简单好用，可以产生如下CSS代码：

```
td { white-space: nowrap}
```

我们只需简单地写成：

```
td { @include nowrap }
```

通过这种方式，你不必记住white-space有连字符。不幸的是，如果你把“nowrap”记成“now rap”（很多人经常如此），就无能为力了。

下面我们再看一个Web设计中与文本相关的常见任务：有图片时，完全不显示文本。

3. 用replace-text将图片转换为文字

尽管有诸如@font-face、Cufón等其他新技术的不断发展，设计师们在必要时还是采用传统的方法来改进Web排版——把文字处理成图片显示出来。常常在头条和其他核心的页面元素上，图片可以传递更复杂的设计元素，有时你无法用标准的字体来处理。在这些情形下，许多设计师都急于在该文本原本的位置放一张<img>了事。对于可访问性（和SEO）而言，更好的建议是用CSS来处理。考虑图4-4中的例子，我们会悄悄利用图片强大的排版魔力进行特殊的渲染，并以此替换头条。



图4-4　替换Coffee文本

为了用CSS实现它，你可以做如下的事情：

```
h1.coffee {
  text-indent: -119988px;
  overflow: hidden;
  text-align: left;
  background-image: url('/images/coffee-header.png');
  background-repeat: no-repeat;
  background-position: 50% 50%;
}
```

第一步是对默认的文字设置一个负值的缩进将其隐藏起来。（如果你有一个超过119,988像素宽的显示器，请和我做朋友吧！）然后通过`background`属性把文本转换成图片。Compass通过`replace-text`辅助器把这件事情变得更简单了：

```
h1.coffee { @include replace-text("coffee-header.png") }
```

不用提供图片的`/images`这部分路径。这是因为Compass内部会使用一个`image-url`的辅助器，它依赖于Compass的配置信息，注明图片的路径。最终你只需要提供图片的文件名即可。第7章将介绍更多关于图片辅助器和其他资源辅助器的信息。

Compass同样提供了一个定制版本的`replace-text`混合器：`replace-text-with-dimensions`，它会根据图片的宽高设定元素的尺寸。

现在你已经掌握了Compass中处理文本的一些实用辅助器。下一节，我们会介绍一些处理一般布局的技巧。

4

4.3 布局辅助工具

除了网格，布局模式估计是样式表中最个性化的部分。Compass提供了一对辅助器来应对不同的布局场景：粘滞的页脚和可伸展元素。在尝试下面的例子之前请先通过`@import "compass/layout";`导入布局模块。

4.3.1 粘滞的页脚

想象一个场景，你需要页脚始终停靠在页面的最下方。你第一想到的可能是`position: fixed`。但不幸的是，如果你需要支持IE6的话，那么CSS不会直接生效。下面我们介绍Ryan Fait的实现方式，以代码清单4-6中的标记为例。

代码清单4-6 粘滞的页脚的标记

```
<body>
  <div id="content">
    页面内容……
    <div id="bump"></div>
  </div>
  <div id="footer">
    停靠在页脚的内容。
  </div>
</body>
```

你可以用下面的CSS粘滞页脚。

代码清单4-7 粘滞的页脚的CSS

```
html, body {
  height: 100%;                    <-- 兼容IE6
}

#content {
  clear: both;
  min-height: 100%;
  height: auto !important;
  height: 100%;
  margin-bottom: -40px;
}
#content #bump {
  height: 40px;
}

#footer {                          <-- 页脚
  clear: both;
  position: relative;
  height: 40px;
}
```

在这个例子中，可以通过`#footer`选择器塑造一个40像素高的页脚，给内容区域设置一个最小的高度为100%，以确保其至少可以占满屏幕的高度。不幸的是，你不得不为`<html>`和`<body>`标记使用`height: 100%`，同时针对IE6设置`height: auto !important`以hack`#content`元素设置的`min-height`。而#bump元素仅仅需要在下方提供足够的内边距以偏移其页脚。

这个CSS不仅考虑到了兼容IE6，而且扩展了更多，它更让你不得不设置3个和页脚高度相关的值。有了Compass的`sticky-footer`混合器，你可以淘汰这种写法了：

```
@include sticky-footer(40px, "#content", "#footer", "#sticky-footer");
```

现在如果你决定创建你的页脚，不论高低，你只需改一个地方，其余的CSS就自动生成了。至此，你已经知道了如何创建一个始终在最下面的页脚，下一节，我们看看如何在父元素内伸展元素。

4.3.2 可伸展元素

流布局被认为是Web界面的核心优势之一，Web设计师也经常理所当然地使用它。而那些有桌面应用开发背景的人，却常常忽视.NET WinForms、JavaSwing、Flash等框架中常见的绝对定位方式。当然，Web通过`positin: absolute`支持了这种布局方式：

```
a.login {
  position: absolute;
  top: 5px; right: 5px; bottom: 5px; left: 5px;
}
```

Compass的`stretch`混合器提供了一个设置这些样式的快捷方式：

```
a.login { @include stretch(5px, 5px, 5px, 5px) }
```

产出的代码和之前相同。stretch混合器有4个参数：$offset-top、$offset-right、$offset-bottom和$offset-left。Compass还提供了只设置一个轴向可拉伸的混合器stretch-x和stretch-y，它们分别只带有$offset-left和$offset-right、$offset-top和$offset-bottom两个参数。

4.4 小结

本章，我们了解了一些Compass省时省力的工具，终于可以告别样式表中枯燥的工作了。我们使用针对性的样式重置在全局样式重置量级过重的情形下，只清除部分元素的样式。看到了如何使用link-colors、hover-link、no-bullet、pretty-bullets和horizontal-list设置链接和列表的样式；也看到了Compass提供哪些处理文本溢出和折行、布局、颜色，甚至清除浮动的快捷方式。

下一章，我们会看到一些Compass的进阶CSS3特性，使创建复杂的用户界面主题变得更容易。

第 5 章 通过Compass使用CSS3

本章内容

- ❑ 用Compass的CSS3模块创建跨浏览器的CSS3样式表
- ❑ 在低版本IE中支持一些CSS3的特性
- ❑ Compass里的CSS3高阶技能

通过之前3章的学习，我们领略到了Compass如何移除流程中重复且枯燥的运算，快速创建样式表。之前，我们的重点一直是适用多年的CSS选择器及其属性上。而在这一章，我们会聚焦更多Web设计的前沿方法，它们被统称为CSS3。

5.1 什么是 CSS3

CSS3，或称第三代层叠样式表，是基于CSS2的规范建立起来的。我们现在称为CSS3的第一份草案诞生于1999年，它包含20多个模块以及很多特性，并在不断发展完善。最近几年，随着浏览器开始支持CSS3，样式表开发者们才真正从中获益。那么，CSS3为我们带来了什么呢？嵌套、变量，还是混合器？不好意思，这些都不是，你仍然需要通过Sass搞定它们。CSS3的变革可以归纳为两部分——更给力的选择器，帮助我们定位元素；各种新的属性，用于修饰元素的外观。我们会用本章剩余的篇幅介绍新属性。

5.1.1 新属性：浏览器前缀让你烦透了吧

尽管我们常说CSS3是一组成型的特性，但实际上这些特性的完善程度不尽相同，有的已经正式通过，有的还处于草案阶段。由于浏览器厂商有各自的发布周期，浏览器接纳新特性的速度也不同，与此同时规范也在快速发展迭代。因此，浏览器厂商通常会先以带有厂商前缀的方式引入CSS3的新特性。设想一个早期通过原生支持的`border-radius`属性实现Web 2.0圆角的例子：

```
button, a.button {
  -webkit-border-radius: 5px;
     -moz-border-radius: 5px;
          border-radius: 5px;
}
```

尽管现在最新的浏览器版本都已经支持不带厂商前缀的border-radius属性，但以前可不是这样。浏览器对圆角的引入分别是从Safari 3.2和Firefox 3.5开始的，并分别带有-webkit-和-moz-前缀。这意味着如果你想支持Safari、Firefox和Opera，你就需要写3个属性。这种复制/粘贴已经使样式表开发工作不轻松了，如果厂商们实现的语法都不一样，那么CSS3属性用起来就真的令人头疼了。

5.1.2　让Compass拯救你

正如第1章里简单介绍过的，Compass承担了繁重的工作，让你从支持每个厂商命名空间的痛苦中解脱出来。你现在可以使用标准的语法，让Compass生成所有的CSS前缀，如代码清单5-1所示。

代码清单5-1　用Compass更快生成厂商命名空间

```
@import "compass/css3";                 <-❶ 导入Compass CSS3支持

.notice {
  @include border-radius(5px);       <-❷ 用border-radius添加圆角
}
```

在项目里，仅仅通过添加Compass的CSS3模块❶，使用border-radius混合器❷，你就可以快速生成支持所有主流浏览器的CSS代码，如代码清单5-2所示。

5

代码清单5-2　用Compass的border-radius混合器生成CSS代码

```
.notice {
  -moz-border-radius: 5px;
  -webkit-border-radius: 5px;
  -o-border-radius: 5px;
  -ms-border-radius: 5px;
  border-radius: 5px;
}
```

正如代码列出来的那样，你不仅支持Safari、Chrome和Firefox，还顺便支持了Opera和IE9。即便如此，如果你并不打算支持所有的浏览器厂商，那么这份代码就显得有额外负担了，那该怎么办呢？Compass提供了一种简单的方式，通过浏览器支持模块设置一些选项，来配置我们希望创建的厂商命名空间，如代码清单5-3所示。

代码清单5-3　在Compass里配置厂商命名空间

```
@import "compass/css3";

$experimental-support-for-opera: false;
$experimental-support-for-microsoft: false;
$experimental-support-for-khtml: false;

.notice {
  @include border-radius(5px);
}
```

Compass提供了几个类似于`experimental-support-for-xxxx`的变量配置项。如果我们以`false`覆盖其默认值，Compass在产出CSS时就会忽略相应的厂商命名空间。

5.1节已经介绍了如何通过Compass解决一些令人头疼的CSS3属性用法。在本章余下的部分，我们将探讨一些使用CSS3创建现代设计元素以及通过Compass快速搞定它们的示例。

5.2　通过 Compass 使用 CSS3

你已经了解了Compass如何将CSS3属性中丑陋的厂商前缀抽离出来。这一节会介绍更多Compass的CSS3模块，以及如何让设计工作事半功倍。

5.2.1　圆角

尽管我们确定，即使网页没有圆角也不影响阅读，但设计师们（也包括管理者和用户）都喜欢圆角。按钮、标签、侧边栏以及表格都无法逃脱被砍掉边角然后磨圆的宿命。设计师们为此运用了很多技术，如多背景图片或借助额外的标签。幸好，CSS3通过`border-radius`属性让这份工作回到了正轨。我们看看图5-1。



图5-1　圆角按钮

在图中，我们看到三个按钮（和一个侧边栏），它们都有美观的圆角。注意，每个按钮的圆角程度，或者说边框半径是不同的。如果你不需要支持4以下版本的Safari、3.6以下版本的Firefox、移动Safari或旧版安卓浏览器的话，就大可放心地使用CSS3里的`border-radius`属性。但是，如果你需要支持它们，那CSS代码就比较乱了，这可要“归功”于厂商命名空间，如代码清单5-4所示。

代码清单5-4　CSS圆角

```
button {
  background: red;
  border: 0;
  color: #fff;
  line-height: 30px;
  width: 100px;
}

button.rounded {
  -moz-border-radius: 5px;
```

这三个按钮的基本样式

Firefox < 3.6

```
  -webkit-border-radius: 5px;           <-- Safari<5、移动版Safari、
}                                            安卓浏览器

button.really-rounded {
  -moz-border-radius: 10px;
  -webkit-border-radius: 10px;
}
button.extreme-rounded {
  -moz-border-radius: 30px;
  -webkit-border-radius: 30px;
}
```

因为老版本的浏览器需要厂商命名空间来支持这一特性，所以你在使用`border-radius`属性时不得不包含`-moz-`和`-webkit-`前缀。这类无聊的工作让CSS看起来更像打字练习而非设计。幸亏Compass有一个混合器可以帮助你兼容多浏览器从而实现圆角。我们来看Compass如何完成这项工作，如代码清单5-5所示。

代码清单5-5　Compass中的CSS3`border-radius`

```
button {                    <-- ❶ 基本的按钮样式
  background: red;
  border: 0;
  color: #fff;
  line-height: 30px;
  width: 100px;
}

button.rounded {
  @include border-radius(5px)             <--
}

button.really-rounded {
  @include border-radius(10px)            <-- ❷ 设置圆角
}

button.extreme-rounded {
  @include border-radius(30px)            <--
}
```

例中的CSS很容易理解。你为三个按钮都设置了一些基本样式❶，然后设置了每个按钮的圆角❷。很明显，这里被`@include`的`border-radius`是一个Sass混合器。我们回到了一行代码设置一个按钮圆角的世界。Compass会在生成的CSS中包含我们需要的这些厂商命名空间（基于我们已经在5.1节讨论过的配置变量）。

现在我们知道了Compass如何处理圆角，接下来我们看看另一个通用设计元素：阴影。

5.2.2　CSS3 阴影

阴影是设计师们的常用技巧，它可以把二维的网页变得有立体感，仿佛元素浮在页面上一样。我们来看一个CSS3阴影的例子，如图5-2所示。

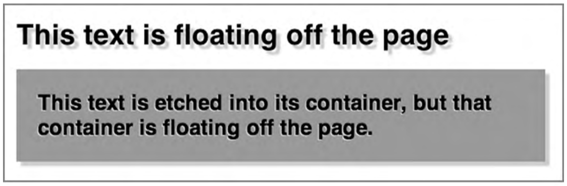


图5-2　CSS3阴影

你在这幅图中能看到多少处阴影呢？如果你说是两处，那么请再仔细看看。你应该看得到3处阴影。其中的两处比较明显：第一段的文字阴影和第二段的盒阴影。如果在此之前，你没有接触过这项技术，那么你很容易忽视第二段文字的镂刻效果。尽管这并不是真正的阴影而是发散出来的亮光，你仍然可以使用CSS3属性实现它。我们看看后面的这个CSS例子，如代码清单5-6所示。

代码清单5-6　用CSS3创建阴影

```
h1 {
  text-shadow: #cccccc 5px 5px 2px;              ❶ 文字阴影
}

h2 {
  -moz-box-shadow: #cccccc 5px 5px 2px;          ❷ 盒阴影
  -webkit-box-shadow: #cccccc 5px 5px 2px;
  box-shadow: #cccccc 5px 5px 2px;
  text-shadow: #dddddd -1px 1px 0;              ❸ 镂刻文字
  background: #999;
  padding: 1em;
}
```

你使用CSS3的`text-shadow`❶和`box-shadow`❷实现了那两个明显的阴影，并再次使用`text-shadow`❸实现了镂刻文字。如果你已经熟悉了本章的套路，那么你会猜到Compass为`box-shadow`提供了混合器来帮助你生成那些厂商命名空间，如代码清单5-7所示。

代码清单5-7　使用Compass的`box-shadow`混合器

```
h2 {
  @include box-shadow(#ccc 5px 5px 2px);
  text-shadow: #ddd -1px 1px 0;
  background: #999;
  padding: 1em;
}
```

Compass再一次通过`box-shadow`混合器解决了烦人的厂商命名空间问题。你可能会惊讶，尽管CSS3中的`text-shadow`属性并不存在厂商命名空间的问题，但Compass还是提供了这样的混合器。这是因为Compass的`box-shadow`和`text-shadow`混合器都可以被应用为多重阴影。我们来看图5-3，它展示了一个运用多重CSS3阴影的例子。

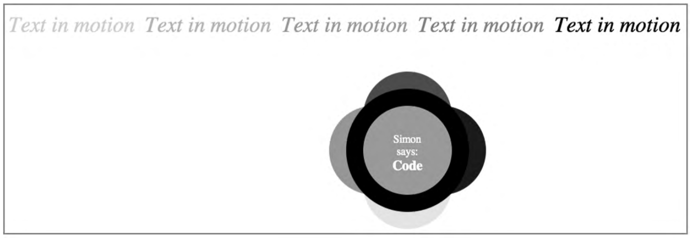


图5-3　为一个元素运用多重CSS3阴影

在这幅图里，你看到了如何运用多重CSS3阴影实现非常复杂的效果。首先，我们设置了动感的文字。然后，我们模拟出“Simon Says”这一80年代的老式游戏。我们看看实现这些效果的CSS代码，如代码清单5-8所示。

代码清单5-8　运用CSS3的多重阴影

```
.motion {
  text-shadow:                              ❶ 动感的文字
    rgba(0, 0, 0, 0.5) -200px 0 0,
    rgba(0, 0, 0, 0.4) -400px 0 0,
    rgba(0, 0, 0, 0.3) -600px 0 0,
    rgba(0, 0, 0, 0.2) -800px 0 0;
  font-size: 2em;
  font-style: italic;
  text-align: right;
}
.simon {
  -moz-border-radius: 100px;
  -webkit-border-radius: 100px;
  border-radius: 100px;
  -moz-box-shadow:
   black 0 0 0 25px,                ❷ 黑色的圆环
    red 0 -50px 0,
    blue 50px 0px 0,                ❸ 多个彩色的按钮
    yellow 0 50px 0,
    lime -50px 0 0;
  -webkit-box-shadow:
    black 0 0 0 25px,
    red 0 -50px 0,
    blue 50px 0px 0,
    yellow 0 50px 0,
    lime -50px 0 0;
  box-shadow:
    black 0 0 0 25px,
    red 0 -50px 0,
    blue 50px 0px 0,
    yellow 0 50px 0,
    lime -50px 0 0;
  background: #999;
  color: #fff;
  height: 50px;
```

5

```
  margin: 100px auto;
  padding: 40px;
  text-align: center;
  width: 50px;
}
```

我们运用多重text-shadow❶横向排布多个阴影，并且令透明度递减，从而实现动感文本的效果。对于“Simon Says”游戏来说，我们运用了一个spread值为25px的黑色阴影❷围绕在游戏的正中间形成一个圆环。然后我们使用了一系列彩色阴影❸并且在x轴和y轴的每个方向分别偏移一点形成其他按钮。如你所愿，Compass去掉了重复的代码，如代码清单5-9所示。

代码清单5-9　Compass中的多重CSS3阴影

```
.motion {
  @include text-shadow(                   <-❶ 多重文字阴影
    rgba(#000,.5) -200px 0 0,
    rgba(#000,.4) -400px 0 0,
    rgba(#000,.3) -600px 0 0,
    rgba(#000,.2) -800px 0 0
  );
  font-size: 2em;
  font-style: italic;
  text-align: right;
}

.simon {
  @include border-radius(100px);
  @include box-shadow(                    <-❷ 多重盒阴影
    black 0 0 0 25px,
    red 0 -50px 0,
    blue 50px -0px 0,
    yellow 0 50px 0,
    lime -50px 0 0
  );
  background: #999;
  color: #fff;
  height: 50px;
  margin: 100px auto;
  padding: 40px;
  text-align: center;
  width: 50px;
}
```

box-shadow混合器❶再次为我们省去了厂商命名空间的工作。乍一看text-shadow混合器❷、❸并没有带来更多的好处，因为我们无需处理厂商命名空间，但是如果我们在这个例子的基础上更进一步，你就能感受到它的威力，如代码清单5-10所示。

代码清单5-10　在Compass中重用文字阴影

```
$shadow-1: rgba(#000,.5) -200px 0 0;     |
$shadow-2: rgba(#000,.4) -400px 0 0;     |-❶ 定义文字阴影
$shadow-3: rgba(#000,.3) -600px 0 0;     |
$shadow-4: rgba(#000,.2) -800px 0 0;     |
```

```
.motion {
  @include text-shadow($shadow-1, $shadow-2, $shadow-3, $shadow-4);   ❷ 使用全部的4个阴影
  font-size: 2em;
  font-style: italic;
  text-align: right;
}

.skipping {
  @include text-shadow($shadow-2, $shadow-4);   ❸ 只使用两个阴影
}
```

在这个例子中，我们再次把多重阴影运用在了两个元素上，但这次，有一个元素只使用了两个阴影。我们可以只定义一次阴影而将其多次传递给`text-shadow`混合器以达到重用的目的。如果你使用Sass脚本和第12章的相关技巧来创建程序化的阴影，那`text-shadow`混合器就变得更有趣了，如代码清单5-11所示。

代码清单5-11　在Compass中使用默认设置创建阴影

```
$shadow-color: #ccc;   ❶ 共享的阴影设置
$shadow-h: 5px;
$shadow-v: 5px;
$shadow-blur: 0;

$default-text-shadow-color: $shadow-color;   ❷ 默认文字阴影
$default-text-shadow-h-offset: $shadow-h;
$default-text-shadow-v-offset: $shadow-v;
$default-text-shadow-blur: $shadow-blur;

$default-box-shadow-color: $shadow-color;   ❸ 默认盒阴影
$default-box-shadow-h-offset: $shadow-h;
$default-box-shadow-v-offset: $shadow-v;
$default-box-shadow-blur: $shadow-blur;

h1, h2 {font-family: sans-serif;}

h1 {
 @include text-shadow;   ❹ 默认值
}

h2 {
  @include box-shadow;
  @include single-text-shadow(#ddd, -1px, 1px);   ❺ 特殊的文字阴影
  background: #999;
  padding: 1em;
}
```

在原始代码里，这些重构并不会带来任何好处。而实际上，它的代码比原本的例子更长了。在做评判之前，让我们一步步分解来看。首先，我们通过一些Sass变量设置了几个共享的阴影设置❶。我们可以重用这些变量来设置Compass默认的`text-shadow`效果❷和`box-shadow`效果❸。现在我们不传递任何参数也可以调用这些混合器了❹。并且，我们还可以通过`single-text-shadow`混合器传递我们希望覆盖的参数来添加其他文字阴影❺。

5

诚然，对于一个只有两三个阴影的简单页面来说，这有点小题大做，但是想象一下拥有大量元素的大规模网站，当它需要统一文字阴影或盒阴影时，这种工作就非常必要了。相对于创建特殊的CSS类来说，你可以设置一些默认值来应对样式表中大量重用的情况。

现在你已经知道Compass是如何处理文字阴影和盒阴影的了，下面让我们来认识一些有难度的东西：CSS3颜色渐变。

5.2.3　颜色渐变

如你在使用`border-radius`、`text-shadow`和`box-shadow`的CSS3例子中看到的，厂商命名空间是一个痛点。你已经知道Compass如何帮你敲入`-webkit-`、`-moz-`和其他前缀。而在对CSS3颜色渐变的支持上，你会发现Compass不仅仅能帮你敲字，还能节省你的脑细胞。来看看图5-4展示的在日常生活中会遇到的例子。

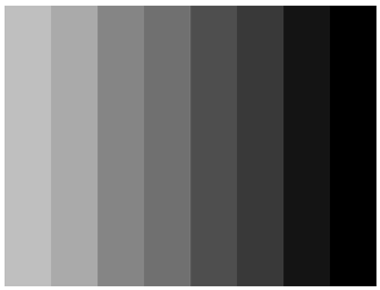


图5-4　电视机测试图案的颜色渐变

我们在电视机没有信号的时候见到过这个熟悉的测试图案。这个图案是8种颜色的横向线性颜色渐变，每种颜色占图案宽度的12.5%且均匀分布。我们试着用CSS在网页上重构这个图案，如代码清单5-12所示。

代码清单5-12　用CSS3实现电视机测试图案

```
#pattern {
  background: -webkit-gradient(          ❶ 旧版Webkit语法
    linear, 360deg, 360deg,
    color-stop(0%, #bfbfbf),
    color-stop(12.5%, #bfbfbf),
    color-stop(12.5%, #bfbf00),
    color-stop(25%, #bfbf00),
    color-stop(25%, #00bfbf),
    color-stop(37.5%, #00bfbf),
    color-stop(37.5%, #bfbf00),
    color-stop(37.5%, #00bf00),
    color-stop(50%, #00bf00),
    color-stop(50%, #bf00bf),
    color-stop(62.5%, #bf00bf),
    color-stop(62.5%, #bf0000),
```

```
    color-stop(75%, #bf0000),
    color-stop(75%, #0000bf),
    color-stop(87.5%, #0000bf),
    color-stop(87.5%, #000000),
    color-stop(100%, #000000));
  background: -webkit-linear-gradient(          ◁—❷ 新版语法
      360deg,
    #bfbfbf 0%, #bfbfbf 12.5%,
    #bfbf00 12.5%, #bfbf00 25%,
    #00bfbf 25%, #00bfbf 37.5%,
    #bfbf00 37.5%, #00bf00 37.5%,
    #00bf00 50%, #bf00bf 50%,
    #bf00bf 62.5%, #bf0000 62.5%,
    #bf0000 75%, #0000bf 75%,
    #0000bf 87.5%, #000000 87.5%,
    #000000 100%);
  /* 省略 -ms、-o、-moz 各自版本的代码 /*
  height: 300px;
  margin: 100px auto;
  width: 400px;
}
```

即使代码清单里已经省略掉了重复的厂商命名空间版本，看起来我们还是为实现这8个竖向条纹写了一大堆CSS代码。因为线性颜色渐变第一次加入CSS3是在2008年，当时只有Safari等基于WebKit内核的浏览器支持，并且还是旧版语法❶。而现在，众多浏览器的最新版本支持的都是简化过的新版语法❷。我们来看看Compass中同样的例子，如代码清单5-13所示。

5

代码清单5-13 通过Compass实现电视机测试图案

```
#pattern {
  @include background(          ◁—❶ 背景模块中的颜色渐变
  linear-gradient(
    360deg,
    #bfbfbf 0%,
    #bfbfbf 12.5%,
    #bfbf00 12.5%,
    #bfbf00 25%,
    #00bfbf 25%,
    #00bfbf 37.5%,
    #bfbf00 37.5%,
    #00bf00 37.5%,
    #00bf00 50%,
    #bf00bf 50%,
    #bf00bf 62.5%,
    #bf0000 62.5%,
    #bf0000 75%,
    #0000bf 75%,
    #0000bf 87.5%,
    #000 87.5%,
    #000 100%));
  height: 300px;
  margin: 100px auto;
  width: 400px;
}
```

原来如此！使用Compass，你只需要在样式表中写4行代码就完成之前那段CSS所做的事情。通过Compass的CSS3模块中的`background`混合器，你就可以使用CSS3语法❶创建线性（或放射性）颜色渐变，并远离命名空间和旧版语法的困扰。它不仅仅让你少打了几个字，更省得你考虑那些语法，让你专注在设计上。

本章进行到这里，我们已经知道Compass如何让CSS3的工作变得更容易更有趣。在本章剩下的篇幅里，我们会深入到一些更高阶但不太常用的CSS3概念当中，并看一看Compass对它们的支持如何。

5.2.4　用`@font-face`嵌入字体

找一些你喜欢的报刊杂志或印刷品。想想看上面的品牌都是怎么印刷出来的。它的重点在于在设计中选择（或调整）和使用字体。不幸的是，对于Web设计师来说，我们一直受困于数得着的几种有限字体，因为我们在设计中使用的字体必须是用户电脑安装并使用的字体。我们常常用字体栈或字体列表来表示浏览器在这个页面的这个元素上应该使用的字体的偏好顺序：

```
font-family: Georgia, "Times New Roman", serif;
```

尽管`@font-face`规则早在CSS2就被引入，但是它只是初步支持了一个IE中专属的格式。最近其他浏览器已经加入了对OTF、WOFF、SVG和TTF的支持，所以我们可以为自己的设计提供字体服务了。但麻烦在于，就像其他CSS3模块一样，浏览器仍然没有在Web字体的格式和CSS用法上达成共识。真庆幸有了Compass。

一个小警告：在通过`@font-face`使用任何字体之前，请检查其许可证，以确认你是否有权利这么做。Font Squirrel和Google Web Fonts都是不错的资源，免费又漂亮，你可以合法地将其嵌入到你的网站中。选中一个字体后，你可以下载一个zip包，里面有所有你为网站提供字体服务所需要的东西，包括用来支持不同浏览器的多个字体文件和与其字体渲染相对应的CSS3样式表。作为演示，我们将标题设置为Font Squirrel的加粗ChunkFive字体，如图5-5所示。



图5-5　非常适合做标题的ChunkFive字体

现在你已经大体知道最终结果了，我们看看如何用CSS3实现它，如代码清单5-14所示。

代码清单5-14　CSS3的ChunkFive标题

```
@font-face {
  font-family: 'ChunkFiveRegular';
  src: url('Chunkfive-webfont.eot');                ❶ IE9

  src: url('Chunkfive-webfont.eot?#iefix')          ❷ IE6至IE8
       format('embedded-opentype'),
```

```
        url('Chunkfive-webfont.woff')                    <-❸ 最新的浏览器
         format('woff'),

        url('Chunkfive-webfont.ttf')              <-❹ Safari等移动浏览器
         format('truetype'),

        url('Chunkfive-webfont.svg#ChunkFiveRegular')  <-
         format('svg');                                 ❺ 低版本的iOS浏览器
  font-weight: normal;
  font-style: normal;
}

h1,h2,h3,h4,h5,h6 {
  font-family: 'ChunkFiveRegular'
}
```

看起来没有CSS新特性是不需要对IE进行hack的，`@font-face`也是如此❶和❷。但是这个特殊的CSS3特性需要针对主流浏览器❸和❹以及一些低版本移动浏览器❺进行更广范的字体格式支持。尽管Font Squirrel在“字体包”里提供了匹配不同字体文件的CSS，这已经很方便了，但是它的样式表假定你在样式表所在的文件夹里提供了这个字体服务。所以，如果你的字体放在了别处（比如第7章将会提到的资源主机），那么你就需要分别处理每个声明过的`url()`地址。在此，Compass再次节省了你的打字时间，它允许你用更少的代码创建相同的CSS，如代码清单5-15所示。

5

代码清单5-15　使用Compass中的`@font-face`

```
@import "compass";
@include font-face("ChunkFiveRegular",
  font-files( "Chunkfive-webfont.woff", woff,
              "Chunkfive-webfont.ttf", ttf,
              "Chunkfive-webfont.svg", svg),
              "Chunkfive-webfont.eot", normal, normal);
```

代码不仅更精简了，也更强大了。这个`font-files`辅助器完成了很多工作。首先，它提供了一个快捷语法，用于创建规则中`url()`和`format()`部分。第二，可能更重要的是，它建立了基于Compass配置的`url()`路径。在开发环境下，它可能是你本地机器的`/fonts`目录；在生产环境中，它可能是http://assets.example.com/fonts。我们会在下一章进一步讨论Compass的这一功能。

5.3　通过 CSS PIE 支持 IE

我们本章讨论的许多特性都能被Firefox和基于WebKit的浏览器很好地支持，但依然有大部分CSS3特性仍未被除IE8及更高版本之外的IE版本支持。假如一家企业的浏览器被长期锁定在老版本的IE上，而你又需要支持IE，在这种情形下你该怎么做呢？坦白地讲，在低版本浏览器里看到一些尖角不是什么大不了的事，就像来自dowebsitesneedtolookexactlythesameineverybrowser.com[1]的截图5-6一样。

---

① 网址的英文意思是：“有必要让网站在每个浏览器里都长得一模一样吗？”——译者注

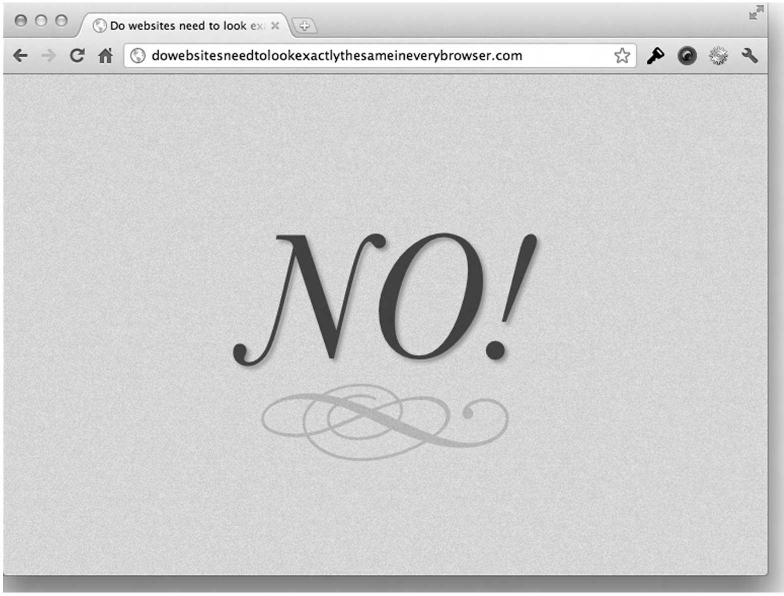


图5-6　有必要让网站在每个浏览器里都长得一模一样吗

但是当网站可以显示得不一样时，Compass为那些不想放弃经典浏览器的用户提供了一个解决方式。CSS3 Progressive Internet Explorer，或CSS3 PIE，是一个由Jason Johnston创建的项目，它针对低版本IE提供很多CSS3特性的polyfill[①]。通过使用专属于低版本IE的HTC behavior，PIE提供了很多CSS3特性的完整或部分支持。包括：

- Border-radius
- Box-shadow
- Border-image
- 多重背景图片
- 线性颜色渐变的背景图

让我们回顾几个5.2节讨论过的CSS3项目，并找到PIE在低版本IE中支持圆角和线性颜色渐变的方法。比如图5-7中的这几个按钮。

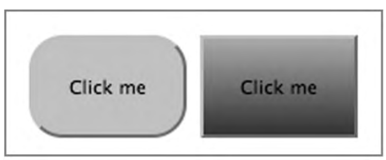


图5-7　简单的圆角按钮和背景颜色渐变的按钮

① polyfill指在Web开发中使某个浏览器本不支持的功能得以实现的代码片段，多用于低版浏览器对HTML5新特性的支持。——译者注

如我们之前在本章看到的，这些按钮的跨浏览器CSS代码虽然简单却冗长，如代码清单5-16所示。

代码清单5-16 用CSS3实现圆角和颜色渐变

```
.rounded {                                    <— CSS3 圆角
  -moz-border-radius: 20px;
  -webkit-border-radius: 20px;
  -o-border-radius: 20px;
  -ms-border-radius: 20px;
  border-radius: 20px;
}

.gradient {                                   <— CSS3 线性颜色渐变
  background: -webkit-gradient(
    linear, 50% 0%, 50% 100%,
    color-stop(0%, #aaaaaa),
    color-stop(100%, #333333));
  background: -webkit-linear-gradient(#aaaaaa, #333333);
  background: -moz-linear-gradient(#aaaaaa, #333333);
  background: -o-linear-gradient(#aaaaaa, #333333);
  background: -ms-linear-gradient(#aaaaaa, #333333);
  -pie-background: linear-gradient(#aaaaaa, #333333);
  background: linear-gradient(#aaaaaa, #333333);
}
```

根据PIE文档，若想把支持范围扩大到低版本IE，你需要做一些小的改动。首先，你需要为网站下载并添加HTC组件。完整的安装流程详见PIE网站。不过现在我们假设你已经把这个文件部署在了`/stylesheets/PIE.htc`上。接下来，你需要在样式表里添加一些额外的规则，如代码清单5-17所示。

代码清单5-17 用CSS3 PIE实现圆角和颜色渐变

```
.rounded, .gradient {                              <—❶ 应用了PIE行为
  behavior: url("/stylesheets/PIE.htc");
  position: relative;
}

…

.gradient {
  background: -webkit-gradient(linear, 50% 0%, 50% 100%,
                               color-stop(0%, #aaaaaa),
                               color-stop(100%, #333333));
  background: -webkit-linear-gradient(#aaaaaa, #333333);
  background: -moz-linear-gradient(#aaaaaa, #333333);
  background: -o-linear-gradient(#aaaaaa, #333333);
  background: -ms-linear-gradient(#aaaaaa, #333333);
  -pie-background: linear-gradient(#aaaaaa, #333333);
  background: linear-gradient(#aaaaaa, #333333);
}
```

为了让PIE介入，你需要应用相关的元素❶，再加上`position: relative`以修复IE的bug。

接下来，为了实现颜色渐变，你需要使用一个非标准的CSS属性-pie-background以告诉PIE为你的按钮添加一个颜色渐变的背景。

如你所料，这些工作（包括安装）在Compass里变得更容易了。你可以通过命令行在项目里引入PIE资源并展开一个有完善文档的示例样式表：

```
compass install compass/pie
```

在这个已经具备PIE样式表和HTC组件的地方，你基于Compass的PIE混合器写一点Sass，便可以创建出之前代码清单里的CSS，如代码清单5-18所示。

代码清单5-18　使用Compass PIE

```
@import "compass/css3/pie";                    ❶ 导入 PIE

.pie-element {
  // 默认就是relative，所以写relative是多余的，这里刻意说明一下。
  @include pie-element(relative);
}

.rounded {
  @include pie;                                ❷ 扩展pie元素的class
  @include border-radius(20px);
}

.gradient {
  @include pie;
  @include background(linear-gradient(#aaa, #333));
}
```

就这么简单！你导入了Compass的PIE模块❶，然后将其应用到你的按钮当中❷。其他的CSS都是我们之前看到过的Compass的CSS3支持。很明显，PIE更适合与Compass搭档。

5.4　小结

在这一章，我们介绍了Compass如何让撰写CSS3更快更有乐趣。你也看到了我们在完全不需要和厂商命名空间打交道的情况下，如何使用Compass的CSS3混合器实现圆角、创建阴影、应用颜色渐变以及完善排版。我们探讨了如何针对浏览器使用配置属性，以及如何使用CSS3 PIE支持低版本的IE。

在下一章，我们会看到Compass如何自动把独立的背景图片替换成一张图片精灵，以增强网站的性能。

Part 3 第三部分

来到生产环境

本书的前两部分介绍了Sass和Compass，及其带来的许多样式表撰写方式的转变。第6章深入探寻Compass在精灵[①]方面的魔力。我们会讲解使用CSS精灵的原因并展示一些简单而高级的精灵用例。你将会看到Compass如何组合并测量你CSS中的图片，完全自动化地处理精灵。同时你也会学到如何配置其布局、空间、位置、类名，等等。

第7章将介绍Compass如何让你从本地开发原型轻松转移到生产环境的网址或Web应用中。你还会知道如何使用Compass的资源辅助器，让样式表中所有URL都随着一个简单的配置项变化而更新。我们还会告诉你当Compass无法找到样式表中引用的图片时如何发出警告，帮你杜绝无效的图片链接。最后我们会提供一些在浏览器中进行设计的建议，以及如何准备你的样式表以便将其发布到生产环境。

第8章帮助你实现样式表的最佳性能。你会学到用`@import`串联样式表，以及如何使用Compass内建的样式表压缩并通过gzip压缩节省下载时间。同时你还会学到Compass如何提供对资源主机的支持来进行跨服务器的分布式下载，也包括对Compass的内联图片和字体的支持以减少HTTP请求数。最终，我们会谈一谈选择器的性能并衡量Sass中过度嵌套选择器带来的性能消耗。

在这部分的最后，你将会得到一张大的结点图，它展示了Sass和Compass如何融入你的开发流程。你将了解如何将一个本地开发环境平稳迁移到生产环境的Web服务器上，你也会有信心展示出样式表的最佳性能。在下一部分，我们会看看Sass的高阶功能，然后我们可以把它们放到一起写成一个Compass扩展并把我们的样式表共享为一个开源项目。

① sprite，指将一个页面涉及的零星图片都包含到一张大图中去。——译者注

第6章　精　灵

本章内容

- ❑ CSS精灵的历史和基本原则
- ❑ Compass混合器让精灵自动化
- ❑ 自定义精灵图片和CSS输出的高阶技巧

在这一章，我们会关注CSS精灵的功能、面临的挑战以及Compass如何把你从Web设计最枯燥的工作中解救出来。

如果你曾经手动创建过精灵，相信你已经乐在其中了！Compass让精灵更加容易。它会创建精灵地图，撰写你不愿意撰写的那些CSS，并将其无缝集成到你的样式表工作流程中。不过这仅仅是一个开始。甚至当你需要Compass对创建精灵地图和生成CSS进行更多控制时，操作起来依然是不可思议的简单。

在这一章，我们会深入研究一个新的Compass项目，所以如果你还没有安装Compass，请移步至附录B寻求帮助。

6.1　精灵的工作原理

在早期，CSS精灵是很简单的。设计师们会把按钮的不同状态做成图片，并把它们做成一张单个图片，如图6-1所示。



图6-1　一个简单的精灵地图示例

然后，在CSS里，他们会设置按钮的宽、高以及背景图片的属性，并在不同的状态下改变背景图的位置，如代码清单6-1所示。

代码清单6-1　简单精灵的CSS

```
.go-button {
  width: 75px; height: 45px;
```

```
  background: url('images/sprite-button-usage.png') top left;
}
.go-button:hover { background-position: center; }
.go-button:active { background-position: right; }
```

按钮的尺寸比这张精灵地图的尺寸小，它创建了一个视口，通过这个视口，可以让不同的图片处于这个可视区域。当用户悬停其上或点击它时，背景图片的位置会发生改变，视口就展示出另一张图（如图6-2所示）。

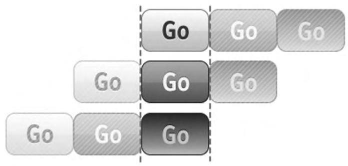


图6-2 CSS创建了一个视口

这是一个早期的简单CSS精灵示例。当然这个例子的未来是非常光明的，因为我们可以通过CSS3样式创建美观的按钮，再也不需要精灵了。

当这项技术第一次流行起来时，大多数的文章都把它视为解决浏览器每次获取下一张图片界面都会闪烁的好方法，仿佛这就是它的优势所在。其实不然，闪烁仅仅是Web性能中看得见的比较明显的一部分，而精灵的使命是解决真正的问题。

6.2 精灵的必要性

从按钮的精灵示例中，你就已经发现精灵可以把3个不同的按钮图形组合成一张单图，同时维持相同的呈现效果。如果把这个技术运用到更深的层面，那么你能构建出一个几乎包括网站中所有背景图片的大图，如图6-3所示。

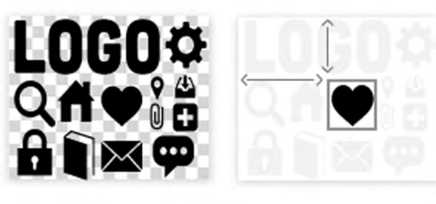


图6-3 一个更加高级的精灵地图

但是为什么要这样做呢？每一张图片都需要在精灵地图里测量、选位，然后排列到你的样式表当中。如果需要重新设计，那么维护起来是非常枯燥费力的。

如果这样做是为了快速下载图片，那为什么不把图片好好压缩一下呢？尽管压缩能起到一点作用，但是不管你信不信，文件大小只是问题的一部分而已。当然，图片压缩得越小，加载速度

越快，这是一定的。但是为了理解精灵真正解决的问题，你需要知道浏览器下载每张图片时究竟做了什么。

6.2.1　HTTP请求越少越好

当在本地创建一个网站时，你的浏览器通常可以直接从硬盘或本地运行的Web服务器请求文件。在这两种情况下，你对请求的感受都是几乎瞬间就成功了，你绝对无法想象远程网络连接时下载文件的痛苦。

每次你的浏览器需要从服务器下载一个文件时，都不得不走完一系列的步骤。我们用最简化的方式把它们列出来：

(1) 浏览器——请求服务器打开一个传输套接字
(2) 服务器——处理请求并作出回应
(3) 浏览器——确认服务器的回应
(4) 浏览器——从服务器请求数据
(5) 服务器——处理请求
(6) 服务器——查询文件
(7) 服务器——初始化文件传输
(8) 浏览器——接受文件传输

你的浏览器即便得到的是很小的一段数据，都会和服务器走完所有流程。所以即使是一张很小的已经压缩过的图片，下载这个图片之前在网络上的琐碎时间开销都是不可避免的。很多现代浏览器尝试着同时下载多个文件以避免为每个文件都重新建立连接。但是即便如此，在网络塞车或类似蜂窝网络的弱网络环境下，问题还是尤为明显。记录一下网站请求的所有JavaScript、CSS文件和图片，很容易发现这个数字在快速增长。

这不仅仅是浏览器端的问题。Web服务器也需要做大量的工作来处理和回应这些请求。主流的网站每秒可能会处理百万级的请求。这意味着你的第一个请求和最后一个请求之间可能隔了数以十万计来自其他用户的请求，这严重拖慢了页面的加载速度。每个额外的请求都会加重Web服务器的负担，降低站点性能，增加成本。

使用CSS精灵可以明显减轻Web服务器的压力。它不仅仅是一个好方法，更是高流量站点的最佳实践和必要措施。但我们要为此付出什么呢？确实，把不同尺寸的图片做成精灵并维护它们的CSS位置需要大量的工作。

6.2.2　手动处理是一种折磨

我们并没有骗你。如果你不得不手动创建并维护很大的精灵地图及其对应的样式表，你可能会疯掉。好吧，也许这有一点极端了，但至少是在给你自己增加负担。

确实，精灵图片能够明显提升网站的加载时间，但是每改变一张图片，你就不得不更新这张精灵地图。如果需要改变图片的尺寸，你就必须移动周围的图片，并引发其他一系列图片位置的

改变，你需要重新测量每张图片，再将其更新到你的样式表中。

很明显这是Web设计师和开发者们想象得到的非常枯燥的工作。所以，尽管应该做，还是有很多人拒绝去做。有些网站不得不采用大规模的精灵也是因为流量高，比如Amazon.com（如图6-4所示）。



图6-4 一张来自Amazon.com的精灵地图

事情可以变得更简单一点吗？只要有人愿意在蔬菜上放点芝士，那么我们都可以从更快更高效的网站中获益。

考虑到这里我们会发现，测量、组合、压缩图片并生成精灵的样式表都是可以通过软件精确自动化完成的。因此，这已经成为近几年的一个热议话题。

在网上简单搜索一下，你就可以找到大量的精灵工具，从命令行到基于浏览器的应用应有尽有。每个工具都提供了不同的功能和不同程度的自动化。其中有些工具是很不错的，但是有一个薄弱环节：和工作流程的集成。在这方面，Compass的方案有着独特的优势。

6.2.3 Compass的方案

因为Compass已经集成到了你的样式表撰写流程中，所以它是生成CSS精灵的理想搭档。你在第7章将会看到，Compass有一个配置文件用来说明网站图片所在的位置，因为Compass会生成你的CSS，所以它是自动化处理精灵的理想搭档。

Compass生成CSS精灵的过程如下：

(1) 让Compass指向一个精灵的文件夹；

(2) 告诉Compass撰写你的精灵CSS；

(3) 编译你的样式表。

通过两行Sass，你就可以告诉Compass根据一个目录下的每张图片生成精灵，测量它们的尺寸，通过每个图片的文件名撰写出不同类名下的背景位置。当你改变图片时，Compass会自动更新你的样式表，生成一个新的精灵并在必要的情况下更新背景位置。这几乎就是魔术啊！

6.3　用 Compass 制作精灵

在这一节，我们会走进一个简单Compass精灵项目。在示例代码中，有一个初始化的项目供你自行跟随学习使用。这个项目已经有了你需要的一切，包括一些来自IcoMoon的免费图标集（https://github.com/Keyamoon/IcoMoon--limited-）和Compass的logo。图6-5展示了我们最初的样子。

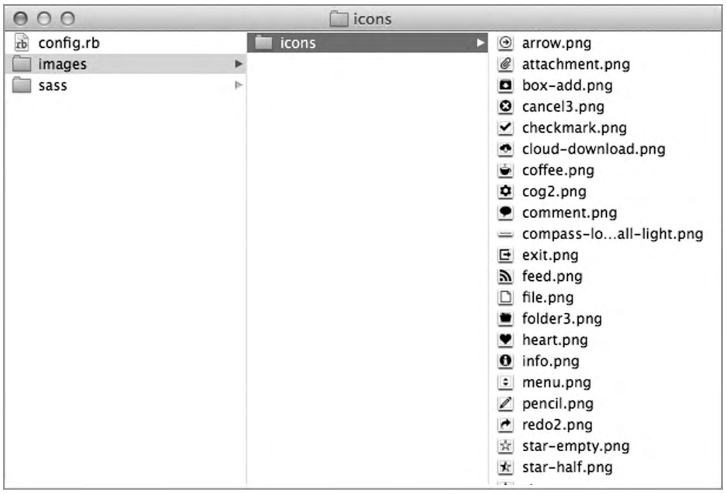


图6-5　示例项目的设置

我们使用PNG格式，因为目前的Compass只生成PNG精灵文件。这没什么大不了的，因为PNG是最理想的精灵格式。现在让我们来看一看它会生成怎样的精灵地图。

6.3.1　创建一个精灵地图

为了将文件夹内的图片转为一张精灵地图，打开screen.scss并加入代码清单6-2所示的代码。

代码清单6-2　用Compass生成一张精灵地图

```
@import "compass/utilities/sprites";
@import "icons/*.png";
```

首先，你导入了Compass的精灵模块。然后，你使用一个精灵导入语句告诉Compass根据`images/icons/`目录下的所有PNG图片生成一张精灵地图。它会在你的图片目录下创建一个类似`icons-s0cad3f8f97.png`名字的图片（如图6-6所示）。



图6-6　生成的精灵地图（部分截图）

默认情况下，精灵是竖向排布的。稍后，我们会学习如何调整精灵的布局、间距，以及如何对单个精灵和整个精灵地图进行设置。但是首先，让我们来看看Compass如何生成精灵的CSS。

6.3.2　生成精灵的CSS

Compass有两个很顺手的混合器，可d为你自动生成精灵的CSS，如代码清单6-3所示。

代码清单6-3　精灵混合器

```
@include all-<map>-sprites;
@include <map>-sprite($name);
```

这里的<map>是一个占位符，它会被替换为包含精灵图片的文件夹的名称。在本例中，它就是`icons`。`all-sprites`混合器会为整个精灵地图撰写所有必要的CSS。而第二个混合器将会输出一个独立命名精灵的CSS。这两个混合器都会随着精灵的导入而创建，因此它们只能在导入之后使用。

我们来看看这两个混合器是如何工作的。你可以跟随学习`automatic-sprites`目录下的`all-sprites-mixin`和`single-sprite-mixin`的代码示例。

1. `all-sprites`混合器

让我们看看Compass中生成所有精灵CSS的工具。找到automatic-sprites/all-sprites-mixin下的例子，如代码清单6-4所示。

代码清单6-4　用Compass生成一个精灵地图

```
@import "compass/utilities/sprites";
@import "icons/*.png";
@include all-icons-sprites;
```

这个`all-icons-sprites`混合器将会为精灵地图中的每个精灵撰写必要的CSS，如代码清单6-5所示。

代码清单6-5　生成的精灵CSS（简化过的）

```
.icons-sprite,
.icons-arrow,
.icons-attachment,
.icons-box-add, ... {
```

```
  background: url('/images/icons-s0cad3f8f97.png') no-repeat;
}

.icons-arrow { background-position: 0 0; }
.icons-attachment { background-position: -16px -96px; }
.icons-box-add { background-position: 0 -64px; }
...
```

为了节省书本的空间，我们简化了前面的CSS输出，实际上它会生成91行代码。现在让我们看看这个混合器都生成了什么：

(1) 它为设置所有来自images/icons/的精灵样式创建了一个基础类`icons-sprite`；

(2) 它为其精灵目录下的每一个文件名创建了类；

(3) 它为所有的精灵都增加了背景图片；

(4) 它为所有的精灵都增加了背景位置。

默认情况下，Compass不会设置这些元素的宽和高。Compass可以自动生成精灵的尺寸，但我们不一定总需要它。

要在项目中使用这个CSS，你可以添加这些精灵类至HTML标签，或者你更愿意使用`@extend`（在第2章里讲解过）从精灵类继承属性，如代码清单6-6所示。

代码清单6-6 使用`@extend`继承精灵的样式

```
.add-button { @extend .icons-box-add; }
```

如果你决定创建不止一个精灵地图，可以添加图片到另一个目录，然后导入它们，再使用`all-sprites`混合器，这样就大功告成了。

2. `single-sprite`混合器

这个混合器将会针对单个的精灵输出CSS。找到automatic-sprites/single-sprites-mixin下的例子，如代码清单6-7所示。

代码清单6-7 使用`single-sprite`混合器

```
@import "compass/utilities/sprites";
@import "icons/*.png";

.add-button {
    @include icons-sprite(box-add);
}
```

这将会输出这个元素样式必要的CSS，如代码清单6-8所示。

代码清单6-8 生成`single-sprite`的CSS

```
.icons-sprite,
.add-button {
  background: url('/images/icons-s0cad3f8f97.png') no-repeat;
}

.add-button { background-position: 0 -358px; }
```

有了single-sprite混合器，就没有必要生成一个类名了，因为样式已经包含在了选择器里。当Compass添加background-image和background-position样式时，会直接使用这些选择器，在这个例子里是.add-button。相比简单方便的all-sprites混合器，这个方式生成更少的CSS，也让你对输出有更多的控制。

这是一个好的开始，但有时你还需要对精灵的过程进行更多的控制。接下来，你将会看到Compass如何让你不费吹灰之力完成更多的控制。

6.4 配置 Compass 精灵

即使对于高阶用户而言，用Compass生成精灵也是非常简单的。还记得那个独特的精灵导入吗？它做得可不仅仅是生成一个精灵。当Compass进行这个独特的导入操作时，首先要确认会影响精灵和CSS生成方式的配置变量列表。而精灵的导入也会因为你在Sass中对精灵的改动而创建一些其他的混合器和函数。

Compass使用包含导入图片的文件夹的名称来命名这些配置变量、混合器和函数。在前面的例子中，你从一个名叫“icons”的文件夹导入了精灵，而这正是all-icons-sprites混合器名字的来源。如果你从嵌套文件夹导入了精灵，那么Compass会使用最末端的包含精灵的文件夹名。所以@import "sprites/social/*.png";会将“social”用于其变量、混合器和函数的名称中。如此一来，即使使用多个精灵地图也不会产生命名冲突。

6.4.1 自定义精灵地图

6

你可以自定义一个精灵地图或通过其配置变量有针对性地定义精灵。影响整个精灵地图的变量的名称以地图名称开头，而仅改变单个精灵的变量的名称是地图名紧跟精灵文件名，如代码清单6-9所示。

代码清单6-9 变量命名表

```
$<map>-<property>: setting;
$<map>-<sprite>-<property>: setting;
```

在我们的示例项目中，精灵文件夹的名称是“icons”，所以改变间距的变量被命名为$icons-spacing。而为了设置icons/attachment.png间距变量，你应该赋值给$icons-attachment-spacing。

记住，这些变量必须在导入精灵之前被定义，否则不会生效。请找到configuring-automatic-sprites文件夹下精灵配置的示例代码。

1. 配置精灵间距

Compass允许你通过配置精灵间距变量改变精灵的间距：

```
$<map>-spacing: 0px;
$<map>-<sprite>-spacing: 0px;
```

默认值是0px，这意味着每个精灵被拼入精灵地图时不带任何间距。设置这个变量或单独给每个精灵赋值，会使得每个精灵在拼合时在周围增加若干像素的透明间距。这段示例代码可以在comfiguring-automatic-sprites/spacing文件夹找到，如代码清单6-10所示。

代码清单6-10　配置精灵间距

```
$icons-spacing: 4px;
$icons-arrow-spacing: 12px;
```

这就意味着你为icons精灵地图中每个精灵设置了4像素的透明间距，并单独为arrow精灵设置12像素的透明间距。图6-7展示了精灵地图的变化。

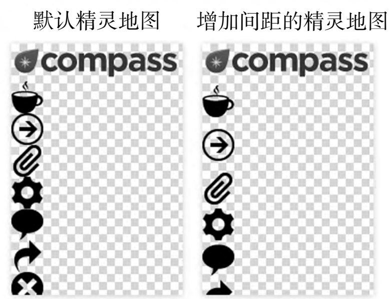


图6-7　精灵地图的间距对照

当小精灵被用在大尺寸元素的背景中时，间距是格外有用的。如果没有间距，小精灵边上的其他精灵可能就会露出来。

2. 设置精灵的重复性

在一些情况下，精灵地图中水平方向的重复精灵会有大用处。为此，你可以设置精灵的重复性变量：

```
$<map>-repeat: no-repeat/repeat-x;
$<map>-<sprite>-repeat: no-repeat/repeat-x;
```

其默认值是no-repeat，但是你可以将其改为repeat-x使其在整个x轴平铺。这一设置可以用在整个精灵地图上，也可以用在某个单独的精灵上。这段示例代码可以在configuring-automatic-sprites/repeat中找到，如代码清单6-11所示。

代码清单6-11　设置精灵的重复性

```
$icons-arrow-repeat: repeat-x;
```

这导致箭头图标横向平铺整个图片。从图6-8中我们可以看到，重复的箭头图标横跨整个精灵地图和最宽的compass的logo对齐。



图6-8　重复的箭头图标

Compass发布时还不支持跨y轴的图片重复。

接下来我们看一看如何设置偏移位置。

3. 设置精灵的位置

有的时候，移动一个精灵的位置是非常有帮助的。Compass允许你通过设置位置变量来横向移动图片：

```
$<map>-position: 0px;
$<map>-<sprite>-position: 0px;
```

这个变量调整了精灵地图中精灵的横向位置。其默认值为`0px`，意味着每个精灵都是左对齐的。这个值可以是百分比，也可以是像素值。下面设置的例子可以在configuring-automatic-sprites/position文件夹中找到，如代码清单6-12所示。

代码清单6-12　设置精灵的位置

```
$icons-position: 4px;
$icons-arrow-position: 100%;
```

在这个例子里，图标精灵地图中的每个精灵都被右移了4个像素，并且箭头精灵移动到了最右边，如图6-9所示。

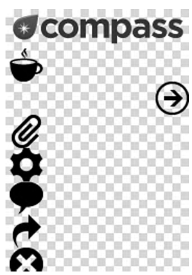


图6-9　精灵位置示例

接下来，我们会讲解完全改变精灵地图布局的方式。

4. 设置精灵地图的布局

Compass有以下几种精灵的布局可供选择：

```
$<map>-layout: vertical/horizontal/diagonal/smart;
```

默认的布局是`vertical`，从整体上影响精灵地图布局，告知Compass如何排布所有的精灵。大多数情况下，你可能需要将布局设置为`smart`，这种布局会让Compass产生最小的空白区域。下面的代码示例可以在configuring-automatic-sprites/layout文件夹中找到，如代码清单6-13所示。

代码清单6-13　智能精灵布局

```
$icons-layout: smart;
```

如果最开始你就在这个项目中使用了智能布局，那么精灵地图看起来会是图6-10的样子。



图6-10　智能布局的精灵地图

位置和重复性的设置只会应用到横向或纵向布局的精灵地图中。对于采用智能布局或对角线布局的精灵地图，位置和重复性的设置是无效的。

接下来，我们会看一看Compass如何移除过期的精灵地图。

5. 清除过期的精灵地图

当添加、删除或改变图片后，会生成新的精灵地图。Compass会自动为你移除旧的精灵地图，或者你也可以把它们保留下来：

```
$<map>-clean-up: true/false;
```

默认情况下，当生成新的精灵地图时，Compass会自动把旧的移除。这会避免你的硬盘被不再使用的文件填满，同时也确保你不会困惑自己的样式表到底在使用哪个文件。如果你喜欢手动移除旧的精灵地图，你也可以将其设置为`false`。

到此为止，我们知道了如何修改Compass生成精灵地图。接下来，我们看看如何自定义Compass生成的CSS代码。

6.4.2　自定义精灵的CSS

你已经知道Compass在精灵地图中布置图片是多么方便。尽管这些精灵地图的改变必然会影响Compass最终生成的CSS，这里还是有一些办法让你直接自定义生成的CSS。

1. 输出精灵的尺寸

如果你想要给一个特殊的精灵设置尺寸，你可以使用精灵尺寸的辅助器：

```
<map>-sprite-height($name)
<map>-sprite-width($name)
```

这里有两个函数可以让Compass测量原始精灵的尺寸并将其输出以便你在样式表里使用它。

通过使用这些辅助器函数，你可以设置单个精灵的宽和高。下面这段代码示例在configuring-automatic-sprites/dimensions文件夹里，如代码清单6-14所示。

代码清单6-14 包含精灵尺寸

```
@import "icons/*.png";
.next {
  @include icons-sprite(arrow);
  width: icons-sprite-width(arrow);
  height: icons-sprite-height(arrow);
}

.add-button {
  @include icons-sprite(box-add);
}
```

另外，如果你想要为精灵地图中的每个精灵自动设置尺寸，你可以为这个精灵地图设置一个配置变量。

```
$<map>-sprite-dimensions: true/false;
```

默认值是`false`，将其设置为`true`会测量每张精灵图片并为其精灵类的宽和高赋值，如代码清单6-15所示。

代码清单6-15 包含精灵尺寸

```
$icons-sprite-dimensions: true;
@import "icons/*.png";
.next { @include icons-sprite(arrow); }
```

产生的CSS如代码清单6-16所示。

代码清单6-16 生成带精灵尺寸的CSS

```
.next {
  background-position: 0 -70px;
  width: 32px;
  height: 32px;
}
.add-button {
  background-position: 0 -358px;
  width: 32px;
  height: 32px;
}
```

相比起手动设置精灵的尺寸，自动生成它们是如此美好。不过如果所有的图标都是相同的尺寸，那么为每一个精灵注明尺寸会让CSS显得臃肿。为此，我们可以为精灵地图的基础类手动设置尺寸。

2. 精灵的基础类

Compass可以方便地通过生成一个基础类为每个精灵应用普通样式。你可以设置基础类变量来选择你的所属类名称：

```
$<map>-sprite-base-class: ".class-name";
```

当你使用全部精灵或单独精灵的混合器时，Compass会输出一个精灵的基础类，并且其选择器还会串联所有设置了`background-image`属性的选择器。每个精灵地图的基础类都以其文件夹的名字命名。如果精灵文件夹的名字是“icons”，则精灵地图的基础类就是`.icons-sprites`。让我们看看你会如何改变它，如代码清单6-17所示。（这个例子在configuring-automatic-sprites/base-class文件夹里。）

代码清单6-17　改变精灵的CSS基础类

```
$icons-sprite-base-class: ".spritey-mcspriterson";
@import "icons/*.png";

.spritey-mcspriterson {
  overflow: hidden;
}
.next {
  @include icons-sprite(arrow);
}
```

每个精灵类都扩展了基础类。所以你添加到基础类的任何样式都会影响你的每一个精灵，如代码清单6-18所示。

代码清单6-18　生成自定义基础类的CSS

```
.spritey-mcspriterson,
.next {
  background: url('/images/icons-s0cad3f8f97.png') no-repeat;
}
.spritey-mcspriterson, .next {
  overflow: hidden;
}
.next {
  background-position: 0 -70px;
}
```

记住，输出的CSS中仅有基础类发生改变。变量名、函数和混合器依然是相同的，它们的名字源于精灵地图的文件夹。

3. 魔术精灵选择器

Compass可以通过伪选择器自动生成精灵的CSS，但是如果需要的话，你可以禁用它：

```
$disable-magic-sprite-selectors: true/false;
```

魔术精灵选择器是默认开启的，也就是说Compass在精灵时会根据以“`_hover`”“`_active`”或`_target`结尾的名字自动输出CSS的`:hover`、`:active`和`:target`伪选择器。这一节的代码在configuring-automatic-sprites/magic-selectors文件夹里。

例如，如果你想要为正常和悬停状态设置不同的精灵，那么就在你的精灵文件夹里加入`arrow.png`和`arrow_hover.png`，Compass就会为悬停的伪类生成CSS精灵的背景，如代码清单6-19所示。

代码清单6-19 生成魔术伪选择器的CSS

```
.next {
  background-position: -32px 0;
}
.next:hover, .next.arrow-hover {
  background-position: -48px -96px;
}
```

如果魔术伪选择器和你选择的图片命名表冲突，请设置为`true`从而对所有精灵地图禁用这个功能。

6.5 驾驭精灵辅助器

至此，我们已经看过了自动生成精灵地图和CSS的工具，以及一些自定义输出的选项。在多数情况下，这些魔术般的混合器可以满足你的所有需求。但是你偶尔也需要抛开这些东西亲自动手。

为了达到生成精灵的目的，Compass依仗了一些辅助器函数。通过对其深入了解，在制作精灵时你会有更大的把握和施展空间。

6.5.1 创建精灵地图

如你早些时候看到的，创建一个特定输入的精灵地图，如`@import "icons/*.png";`，并不仅仅是创建了一个精灵地图；它还为精灵地图和每个精灵设置了混合器和变量。如果你使用精灵辅助器，就不必使用这些变量和混合器，否则会造成精灵导入的大材小用。取而代之的是`sprite-map`辅助器。来看一看`manual-sprites/sprite-helper`的代码示例：


```
sprite-map($glob, ...)
```

这个辅助器接收了一组类似`"icons/*.png"`的图片，以及设置精灵地图或单独精灵的可选关键字参数。这里有几个例子，如代码清单6-20所示。

代码清单6-20 `sprite-map`辅助器

```
$icons: sprite-map("icons/*.png", $layout: smart);
```

它会创建一个智能布局的精灵地图，并把精灵地图的图片URL赋值给`$icons`变量。我们稍后会在其他的辅助器里用这个变量生成CSS，如代码清单6-21所示。

代码清单6-21 `sprite-map`辅助器——设置一个单独的精灵

```
$icons: sprite-map("icons/*.png", $arrow-spacing: 5px);
```

任何精灵地图或单独精灵的属性都可以配置，仅使用我们早些时候提到的去掉<map>命名空间的配置变量。你可以使用`$repeat`替代`$<map>-repeat`的重复性配置；也可以使用`$arrow-repeat`替代`$<map>-<sprite>-repeat`，这里的“arrow”是你配置的精灵的名称。

6.5.2　撰写精灵的CSS

在Compass为你生成精灵地图之后，你仍然需要写出每个精灵的CSS。为此，我们将转向一些其他的辅助器和混合器。

1. **sprite**辅助器

sprite辅助器使得撰写精灵的CSS非常简单：

```
sprite($map, $sprite, [$offset-x], [$offset-y])
```

sprite辅助器需要精灵地图、精灵的名字以及可选的偏移坐标，如代码清单6-22所示。

代码清单6-22　sprite辅助器

```
$icons: sprite-map("icons/*.png");
.next {
  background: sprite($icons, arrow) no-repeat;
}
.add-button {
  background: sprite($icons, box-add) no-repeat;
}
```

这仅仅会输出背景属性，而不会成为一个精灵的基础类或其他任何你不需要的CSS，如代码清单6-23所示。

代码清单6-23　sprite辅助器的CSS

```
.next {
  background: url('/images/icons-s943de15a54.png') 0 -70px no-repeat;
}
.add-button {
  background: url('/images/icons-s943de15a54.png') 0 -358px no-repeat;
}
```

精灵基础类的一个优点是你只需赋值一次背景图片；在这里它会赋值给每一个类，而这是不必要的重复。

2. 设置精灵的位置

为了移除重复的背景图片，你可以用sprite-position辅助器或sprite-background-position混合器取代sprite辅助器。示例代码存放在manual-sprites/sprite-position文件夹：

```
sprite-position($map, $sprite, [$offset-x], [$offset-y])
sprite-background-position($map, $sprite, [$offset-x], [$offset-y])
```

这里的辅助器和混合器都需要一个精灵地图、精灵的名称以及可选的位置偏移量，如代码清单6-24所示。

代码清单6-24　设置精灵的位置

```
$icons: sprite-map("icons/*.png");
.sprite-base { background: $icons no-repeat; }
.next {
```

```
  @extend .sprite-base;
  background-position: sprite-position($icons, arrow);
}
.add-button {
  @extend .sprite-base;
  @include sprite-background-position($icons, box-add);
}
```

sprite-position辅助器和sprite-background-position混合器都做了相同的工作，至于用哪个是个人偏好问题，如代码清单6-25所示。

代码清单6-25 设置精灵位置的CSS

```
.sprite-base, .next, .add-button {
  background: url('/images/icons-s943de15a54.png') no-repeat;
}

.next { background-position: 0 -70px; }
.add-button { background-position: 0 -358px; }
```

在这里你可以看到CSS是如此简单。你有更大的灵活度，且不必重复。不过最好能再加入精灵的尺寸。

3. 设置精灵的尺寸

为包含精灵的尺寸信息，你可以使用sprite-dimensions混合器，它需要精灵地图和精灵的名字,并输出经过测量的尺寸。该混合器的代码示例在manual-sprites/sprite-dimensios`文件夹中，如代码清单6-26所示。

代码清单6-26 sprite-dimensions混合器

6

```
$icons: sprite-map("icons/*.png");
.sprite-base { background: $icons no-repeat; }
.next {
  @extend .sprite-base;
  @include sprite-background-position($icons, arrow);
  @include sprite-dimensions($icons, arrow);
}
```

这个辅助器会测量精灵并在生成的CSS中写明其宽和高的属性，如代码清单6-27所示。

代码清单6-27 sprite-dimensions混合器输出的CSS

```
.sprite-base, .next {
  background: url('/images/icons-s943de15a54.png') no-repeat;
}

.next {
  background-position: 0 -70px;
  height: 32px;
  width: 32px;
}
```

以上就是所有的内容了。有了这些功能和灵活性，你应该不再担心在Web设计项目中加入精灵了吧。

6.6　小结

在本章，我们看到了精灵的由来以及它们在Web设计中的重要作用。我们了解了从远程服务器加载大量图片对性能的影响以及精灵如何作为重要方法解决高流量网站问题。我们还看到了Compass如何完全自动化处理精灵，并探索了配置及控制Compass生成精灵地图和CSS的几种方式。

在下一章，我们会看到一些高阶的Compass特性，它们会帮助你做好生产环境的准备工作。

第7章 从原型到产品

本章内容

- ❑ 生成资源URL的最佳实践
- ❑ 撰写无需Web服务器的样式表
- ❑ 在浏览器中进行设计的技巧
- ❑ 如何为产品编译并构造样式表

网站可以如此简单，一个小孩用一个编辑器和主机账号，在几分钟之内就能轻松建立一个网站。当然了，网站也可以很复杂，拥有动态的内容，每天需要承载上百万的访客。很负责任地讲，Web是涉及专业软件技术领域最广阔的地方之一。从你的中学主页到Google，Web可以在每一层进行优化。

Web之所以复杂并不仅仅因为HTML标记。它的复杂性更多取决于其重度依赖的外部资源，比如图片、其他CSS文件、字体以及样式表，它们都成为沉重的维护负担。

在本章，你会学到如何利用Compass辅助器和配置来生成你的资源URL，使其方便地在原型和产品之间转换。通过使用这些辅助器函数，你可以自由地撰写样式表和HTML而无需Web服务器，这省去了很多令人头疼的工作。同时，你也是在为下一章提到的性能优化措施搭建舞台。

有了CSS3的改进以及Sass和Compass的强大理念，现在是时候重新评估你的设计和原型的工具链了。我们的讲解会涵盖一些在浏览器中进行设计的基本技巧，从而使你判断，从长期来看，Photoshop之类的工具是否拖慢了你的节奏。

在CSS里，你已经习惯于通过搜索&替换技巧使样式表由原型转变到产品。在第3章，你学过了有关Compass的一些描述资源存放位置和服务方式的配置选项。在本章，你会学到如何使用这些简单的配置选项外加一些撰写上的最佳实践消除产品部署时的疑虑，同时允许你简化开发环境。你还会看到如何处理版权提示、源码控制之类的无聊问题。

然后，在下一章，你会学到性能优化的相关内容，从增强样式表性能的每处细节出发，比如通过压缩和图片内联等。但是最先要做的事情是，让我们看看Compass生成URL和得出原型的最佳实践。

7.1　绝对 URL

你的图片都在哪里？这听起来像一个愚蠢的问题。你的图片不就在你的项目文件夹里么？是的，你很容易找到它们，因为你是人类。如果你是一款Web浏览器，找到它们可能就费点劲了。这些图片从你项目的图片文件夹里开始了它的一生，但是它们待会儿可能会被打包、复制、部署、解压、URL重写、压缩、缓存，最终服务到互联网的一个或更多个地方去。

对于一个项目来说，频繁改变图片的存放位置和存储方式是难免的。当你使用Compass时，你将会发现生成和改变图片变得更容易了，但它的优点还不仅限于此。Compass为你生成URL的同时，也确认了图片是真实存在的，且避免浏览器缓存旧的图片。

7.1.1　生成URL资源

CSS提供了`url`函数用来解码URL：

```
background-image: url('/logo.png');
```

URL标识了资源在互联网上的位置，但是当你对照自己的资源时，你常常使用相对URL，且浏览器会根据其对当前请求的了解补齐别的信息。在进入后续内容之前，让我们回顾一些URL相关术语，如图7-1所示。

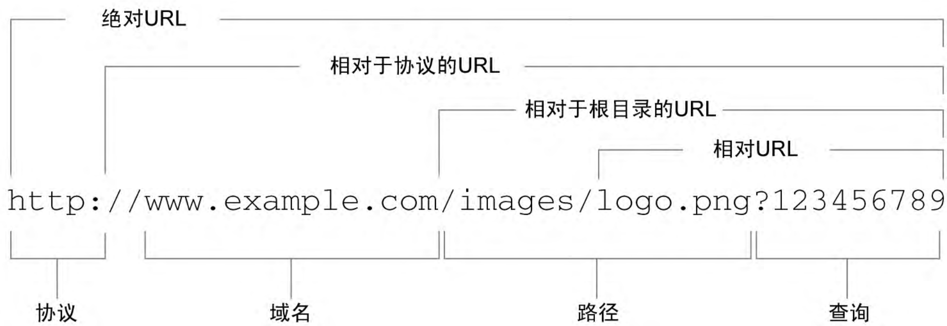


图7-1　分解URL

在CSS中，根据可省略的部分，URL可以按四种方式进行设定，如表7-1所示。

表7-1　四种类型的URL

示　　例	类　　型	描　　述
`url('http://www.example.com/logo.png')`	绝对URL	与请求来源无关
`url('logo.png')`	相对URL	浏览器相对于请求的来源补齐URL，这里的来源指的是CSS样式表，而不是网页。所以如果样式表是http://www.example.com/stylesheets/application.css，那么这个URL会指向http://www.example.com/stylesheets/logo.png

（续）

示 例	类 型	描 述
`url('/logo.png')`	相对于根目录的URL	浏览器相对于CSS样式表的协议和域名补齐URL。所以如果样式表是http://www.example.com/stylesheets/application.css，那么这个URL会指向http://www.example.com/logo.png
`url('//www.example.com/logo.png')`	相对于协议的URL	浏览器通过指定的域名补齐URL，但采用和CSS样式表原始请求相同的协议。当资源服务来自于和你域名不同的网站时，这类URL尤其管用。所以如果样式表是https://www.example.com/stylesheets/application.css，那么这个URL会指向https://imgs.example.com/logo.png

虽然在Sass里你可以继续使用其中任何类型的URL，但Compass的最佳实践是通过资源辅助器函数（http://compass-style.org/reference/compass/helpers/urls/）引用你的资源。Compass提供了三个资源辅助器，每个辅助器都需要你传入相对于资源类的目录的路径（千万不要相对于样式表）：

- ❑ `image-url('logo.png')`——引用保存于图片根目录的`logo.png`文件；
- ❑ `font-url('arial.ttf')`——引用保存于字体根目录的`arial.ttf`文件；
- ❑ `stylesheet-url('randomfile.xml')`——引用保存于`css`根目录的`randomfile.xml`文件。

你可能会注意到这里没有JavaScript的URL辅助器。JavaScript配置选项是存在的，因此Compass扩展可以提供JavaScript文件作为它们安装程序的一部分。同理，这里没有针对`sass`目录的URL生成器，因为这是开发环境中的东西；`stylesheet_url`将会指向你的CSS文件存在的地方。

Compas之所以选择这样的方式是因为它让导入和提炼工作变得更简单。如你即将看到的，用这种单一的语法可以生成CSS支持的所有四类URL。

你的项目配置告诉Compass在哪里可以找到你的资源，所以你可以不必关心如何在样式表中引用它们，并且把样式表从原型到产品产生的改变限定在了配置当中。

稍等，还有别的！Compass还会在每次编译时确认你的URL是否有效！

7.1.2 避免出现死链

你是人类，是人就会犯错。可能是错字，也可能是图片重命名后忘了引用。这些都是在所难免的，不必过分苛责自己。当你使用`image-url($path)`辅助器函数引用一张图片时，Compass会确认这个文件是否存在；如果不存在，则会在编译时从命令行打印出一个警告。同理，使用`font-url($path)`辅助器找不到字体文件时也会有警告出现。

观察下面这个找不到图片的Compass编译输出。在你的命令行里，`WARNING`这一行应该是红色的：

```
[~/Projects/my_compass_project] compass compile
directory stylesheets/
```

```
   create stylesheets/ie.css
WARNING: 'missing.png' was not found (or cannot be read) in images/
     create stylesheets/main.css
     create stylesheets/print.css
```

当然，Compass实际上并不能保证你不会有损坏的链接。配置错误或编译后发生的改变还是会破坏你的站点，但是这项简单的检查已经为苦于调试开发时的常见错误(比如图片显示不出来)的你节省了不少时间。

说到开发时的常见错误，你是否曾经花费十多分钟去排查图片没有正确显示的原因，最后却发现是浏览器缓存的问题？我们在开始使用Compass生成URL之前都有过这种经历吧。接下来我们看看为什么。

7.1.3　通过缓存清理避免旧图片

在开发和发布产品阶段容易遇到的另一个问题是浏览器太懒惰，它们并不喜欢下载东西。这可能对它们来说太乏味了。所以为了重复利用，它们会将图片以及其他资源缓存起来，这是很常见的。对于用户来说这很棒，它提升了浏览器的体验效果。但对于Web开发者来说，这很痛苦。你改变了一张图片，但那些近期下载过它的用户却不会看到。很可惜，因为新的图片比旧图清晰很多。为了解决这方面的问题，Compass在每个资源末端增加了修改时间，并将其作为查询参数。这样你的Web服务器就会正常工作了，当查询参数改变时，图片会被浏览器重新请求。

举个例子：

```
#logo { background-image: image-url('logo.png'); }
```

会被编译成为：

```
#logo { background-image: url('/images/logo.png?1298578273'); }
```

如果你觉得将时间戳作为缓存清理的参数并不是一个好方案，那么你也可以通过配置把它改成别的。比如，发布计数或者文件在源码控制系统中的版本号。完成这些工作需要你写一点Ruby代码。举个例子，在你的Compass配置文件里写入这些内容：

```
# Increment the deploy_version before every
# release to force cache busting.
asset_cache_buster do |http_path, real_path|
  "v=1"
end
```

会使Compass为你的logo图片生成下面这段代码：

```
#logo { background-image: url('/images/logo.png?v=1'); }
```

使用一个查询参数作为缓存清理的策略可能会干扰一些缓存资源的代理功能。(这样的查询字符串会让其误以为该资源是动态生成的。)如果这对你来说是一个问题，你也可以通过一行Compass配置禁用它：

```
asset_cache_buster: none
```

但是，如果你想要最大可能缓存你的资源并且又可以清理，那么最好的办法就是重写资源的路径。通过一些相关的Web服务器配置，你可以生成类似这样的URL：

```
#logo { background-image: url('/images/logo-1307943914.png'); }
```

你需要配置你的Web服务器，这样它会知道如何把时间戳路径对应到真实路径。具体做法取决于你的Web服务器，但是Compass配置看上去像下面这样，如代码清单7-1所示。

代码清单7-1 定义一个基于路径的资源缓存清理

```
asset_cache_buster do |path, real_path|
  if File.exists?(real_path)          # 始终确认文件是否存在

    pathname = Pathname.new(path)     # Pathname比字符串更易于操作

    modified_time = File.mtime(real_path)   # 上次更改时间

    new_path = "%s/%s-%s%s" % [       # 根据四个字符串构造新的路径
      pathname.dirname,
      pathname.basename(pathname.extname),
      modified_time.strftime("%s"),
      pathname.extname
    ]
    {:path => new_path}               # 基于路径的资源的特殊返回格式
  end
end
```

如你所见，任何复杂的逻辑都可以在Compass配置文件中使用，因为这是一段Ruby脚本。比如有些用户为每个资源集成了内容分发网络（CDN）或生成MD5指纹。但是更复杂的逻辑并没有涵盖在本书中，所以如果你想为创建自定义代码寻求帮助，请关注Compass的邮件列表（http://groups.google.com/group/compass-users）或你附近的Ruby专家。

不过在顾虑CDN这么长远的问题之前，你需要先做出原型。CSS最佳实践认为你应该在写第一个选择器之前设置好一个Web服务器。让我们看看为什么使用Sass和Compass时就没有这个必要。

7.2 用 Sass 和 Compass 做原型

不论你是基于一个新概念还是用一些新鲜的模型或框架开始新项目，在项目的生命周期中，Sass和Compass真正大放异彩的时间点都是项目的最开始。有人会认为Sass和Compass是针对大型站点和海量CSS的，但再大的网站也是从小网站发展起来的。

另外，在项目的最开始阶段，所有的东西都具有不确定性，这时Sass和Compass优秀的管理能力就体现出来了。CSS3模块和网格系统让设计工作在浏览器中比在Photoshop中更容易。用Sass的颜色函数体验网站的配色方案，轻松得就像玩一样。

当然，创建原型也需要撰写HTML，但是Sass和Compass并没有涉及HTML的相关业务，所以我们鼓励你学习一款能够快速搭建原型的框架，比如Serve（http://get-serve.com）或Middleman

（http://middlemanapp.com），它们包含了对Sass和Compass的外围支持。

为了在开发阶段用相对于根目录的URL撰写CSS文件，开启一个新项目通常意味着要设置Web服务器、编辑配置文件并修改DNS的host文件。我们来探讨一下Compass如何让你的开发环境变得更简单。

7.2.1　简化你的开发环境

对于你的新站点或新应用来说，相对于样式表的URL是一个不错的起点。它们可以脱离Web服务器工作，所以你在创建原型时可以使用简单的旧的HTML，只要你的样式表及其资源来自相同的域名，它们就可以在任何服务器环境下正常工作。对很多用户来说，这就是他们自始至终的需求。

当然，这种相对URL也面临一个挑战：对CSS开发者来说，你的样式表变得很难分辨。考虑图7-2所示的项目结构。

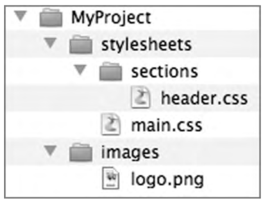


图7-2　一个简单的项目

当使用相对URL时，如果你想从`main.css`移动一些样式到`header.css`，那么你不得不把

```
#logo { background-image: url(../images/logo.png); }
```

改成

```
#logo { background-image: url(../../images/logo.png); }
```

出于这个原因，从创建原型时就始终使用简单的Web服务器和相对于域名的路径的做法一直被认为是最佳实践。甚至当你已经在开发环境有了现成的Web服务器时，开启一个新的站点也并非总是那么容易。在迷宫一样的配置文件、本地host名和服务器端口之间穿梭，这对于很多前端开发者和设计师来说比使用命令行还恐怖。

但因为Compass，你再一次被解放了，你可以通过启用相对资源来使用本地文件。为了使用相对资源，只需要在Compass配置中添加下面这行代码：

```
relative_assets = true
```

当它启用时，不管使用`image-url($path)`、`font-url($path)`还是`stylesheet-url($path)`，Compass都将生成相对路径。值得注意的是，生成的路径是相对于编译好的样式表的（而不是Sass样式表）。意思是说，如果你在多个不同的CSS文件中有共享的部分，则图片引用的相对路径在所有的情形下都会准确无误。举个例子，考虑一下图7-3中展示的项目结构。

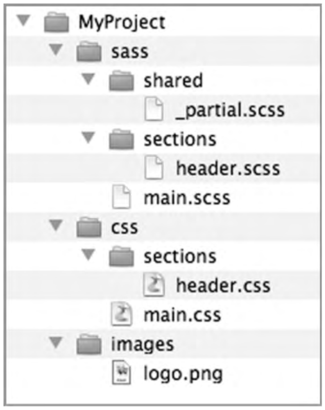


图7-3 一个有共享部分的项目

如果 _partial.scss同时被导入了main.scss和header.scss，且_partial.scss包含：

```
#logo { background-image: image-url("logo.png"); }
```

则生成的main.css会包含：

```
#logo { background-image: url(../images/logo.png?1298578273); }
```

且生成的header.css会包含：

```
#logo { background-image: url(../../images/logo.png?1298578273); }
```

现在你可以从样式表中提炼出核心的内容。你的相对URL将会始终正常工作。有了Compass，你可以快速建立起开发环境。如果你的下一步打算是打开Photoshop，那么请停下来看看下面这节内容吧。

7.2.2 直接在浏览器里设计

如果你是一个自己撰写样式表的设计师，过去你可能先用Adobe的Photoshop或Fireworks为网站建模，然后当你对设计大体满意时才建立样式表。在CSS3和渐进增强之前，需要创建图片切片，于是几乎必须遵循这一工作流程。但是有了CSS3、Sass和Compass后，在浏览器中直接设计网站变得非常高效。

从渐变色到阴影、圆角元素到美观字体，如今的浏览器完全可以摆脱图片处理惯常的设计工作了。在很多情况下，Compass的CSS3模块让撰写代码比在Photoshop里评估效果更加轻松容易。当你完成设计之后，如果真的需要支持老版浏览器，也可以通过页面截图把需要的图片切割下来。

效率的提升不仅仅在体现在只需一次操作。当直接在浏览器上设计时，你能更清晰明确地把设计元素直接串联起来。在整个创建过程中，你不需要思考*这个设计需要的是一些*`#4F9942`。你需要思考，*我需要加重这个标题的颜色*等此类问题。比较便捷的做法是撰写`adjust-color($header-color, $saturation: -15%, $lightness: -25%)`，然后把参数微调几次直至满意为止。但是当你从模型撰写CSS时，最简单的事情是拿出取色器然后复制结果到你的样式表，

这会丢掉你在脑中设计时每个步骤的所有信息并且修改起来也会更困难。

“浏览器存在着各种各样的局限。我们一次次挑战浏览器可能达到的极限固然很好，但大多数情况下你只是为了完成工作。在浏览器中设计允许你在这些局限下工作并乐于拥抱它们，把它们视为设计流程的一部分。比如对于浮动元素而言，我们很难想象和模拟出它在浏览器窗口大小发生变化时的反应。但是你在浏览器里设计时，这很容易就看得出来，也便于测试。

确实，你会发现这个建议让自己在构建初始化设计的过程中多花了一些时间，但是除非你把整件事抛开，否则在浏览器里设计和创建原型就是最有可能缔造高质量产品，并且耗时更短，将来也更易于更改的方式。

现在你已经有一款自己喜欢的设计了，是时候分享给全世界了。我们把它发布成产品吧！

7.3　发布成产品

将CSS发布成产品实际上比发布Sass文件更容易。毕竟CSS是一个简单的文件，所以你只需要把文件复制到你的Web服务器然后就可以收工了。有了Sass和Compass，就多了编译的步骤，产品配置也要做好。虽然一开始这可能有点麻烦，但你很快就会意识到Compass会让你为CSS思考并做你*应该*一直在做的事情。

7.3.1　想不到吧！该挪窝了

你被客户告知，你建立的原计划发布在http://example.com/fancy-app/的应用现在要重命名为http://example.com/super-fancy-app/。客户感觉糟透了，他们认为一切都会顺利通过，但是CEO*竟然*希望这是一个超级应用。最终他们被*这个问题*难住了：“那么这个改动需要多少时间？”

毫不夸张地说，在你的样式表里有上百个URL。如果你得手动修改并测试所有的URL，估计至少需要一个小时的体力劳动加测试。但现在你是Compass用户了，它用到了提供的所有URL辅助器，所以你知道这只是个简单的配置改动。你只需在配置文件里修改一行代码：

```
http_path = "/fancy-app"
```

将其改成下面的样子，然后重新编译：

```
http_path = "/super-fancy-app"
```

不过你现在有个小困扰。你是否告诉他们真实的时间估计结果？我们倾向于告诉他们这仅仅是个小意思，不会产生额外工作量，因为我们喜欢获得肯定并阐明我们在使用样式表编译器时就已经考虑到了。让这些机器替你完成所有的工作岂不是很爽吗？在下一节，你将会看到Compass在部署时如何针对生产环境让工作变得更简单。

7.3.2　为生产环境编译

在Compass中，你可以通过两种不同模式来管理自己的样式表，即开发时和服务于产品时。通常情况下，Compass使用的是`development`环境。如果想使用`production`环境，你需要像按

如下方式编译你的样式表：

```
compass compile --force -e production
```

Compass默认使用产品样式表。你的输出将会被压缩，且大部分的注释会被去掉。我们也可以为当前模式设置一个特殊的配置。比如，如果你想在生产环境下以紧凑型输出，你可以在Compass配置中添加这个条件设置：

```
if environment == :production
  output_style = :compact
end
```

使用这个预设，就有可能利用环境的切换来设置不同的资源配置，当然，这取决于你是在开发机器还是生产环境。

7.3.3 生成相对于域名的资源

默认情况下，Compass生成相对于域名的URL以假定你在通过一台Web服务器浏览网页。现在你将要发布自己的网站了，你需要考虑一些Compass的配置设置以正确生成URL。第一个要考虑的设置是项目的`http_path`——这个默认值是`/`，但是如果你的站点承载在一个子目录中，你就需要改变这个Compass配置：

```
http_path = '/my-app'
```

如果你已经启用了相对资源，你应该禁用它们，因为你不想生成相对于CSS文件的URL，而`relative_assets`设置会优先：

```
relative_assets = false
```

再次编译之后，logo的URL变成了：

```
#logo {
        background: url('/my-app/images/logo.png?1240702589');
      }
```


设想一下，在部署时你的图片被复制到了一个相对于站点根目录的叫做`imgs`的文件夹。这种情况下，你需要为你的项目设置`http_images_dir`：

```
http_path = '/my-app'
      relative_assets = false
      images_dir = 'images' #locally it's the images folder
      http_images_dir = 'imgs' #on the webserver it's different
```

再次编译之后，logo的URL变成了：

```
#logo {
  background: url('/my-app/imgs/logo.png?1240702589');
}
```

但如果你的网站很奇葩，它决定把它的图片放到一个完全不同于HTML的地方，那么你就要转而设置`http_images_path`：

```
http_path = '/my-app'
relative_assets = false
images_dir = 'images'                          <—— 本地图片的位置
http_images_path = '/somewhere-else/imgs'
```

再次编译之后，logo的URL变成了：

```
#logo {
      background: url('/somewhere-else/imgs/logo.png?1240702589');
}
```

Compass有一大堆和资源相关的可能的配置选项。它们通常很容易被忽略，但是如果需要，你能很快和它们成为好朋友。在你的网站上线之前，还有一件重要的事情，就是给i点上点儿，给t划上横线[①]，注意像版权提示之类的可能漏掉的小事情。

7.3.4 添加版权提示

有些站点选择为它们的样式表注明版权。如果你也要这样做，你可能会因为Sass在压缩样式表时去掉了CSS的注释而郁闷。为了解决这个问题，Sass提供了*大声的注释*。大声的注释以一个感叹号标记开始，紧跟在一段CSS注释的星号后面：

```
/*!
  Copyright © 2012, Example Inc. All Rights Reserved.
*/
```

这里值得注意的是，大声的注释会识别Sass脚本，所以它们可以将你的版权提示设置为变量并在站点之中重用：

```
$copyright-year: unquote("2012");
$company-name: unquote("Example, Inc.");
/*!
  Copyright © #{$copyright-year}, #{$company-name}
  All Rights Reserved.
*/
```

这让律师倍感开心不是吗？现在是时候把你的站点放到生产环境中了！

7.3.5 发布CSS很简单

如果你的CSS发布系统是复制样式表到你的Web服务器，那么这件事对你来说不会改变太多。当你为生产环境重新编译你的样式表之后，你还是得到了普通的原来的CSS文件。从这一点出发，你的流程和之前没有区别。关键在于你从服务于用户的地方得到了你的样式表。在图7-4中，你可以看到一个简单的发布流程。

① 英语习语，因为英文的书写习惯是字母i上的点儿和字母t上的横线一开始先不写，最后再统一补齐，这里暗指善始敬终。——译者注

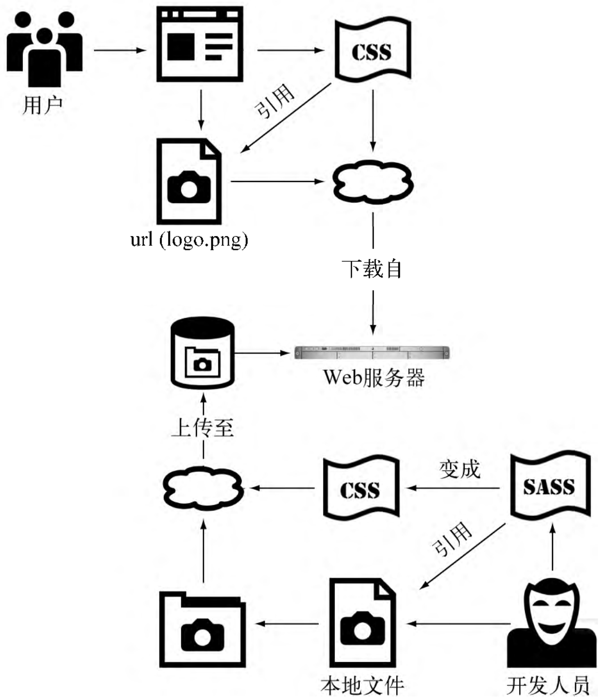


图7-4 Sass样式表如何提供服务

如果你的网站很简单，而你是唯一的开发人员，这个流程看上去很不错。但是如果你在一个团队中工作并且使用了源码控制、构建及（或）发布脚本，那么你需要考虑的就更多了。

7.3.6 跟源码控制、发布流程配合在一起

源码控制最佳实践表明非手工编辑的生成文件不应该出现在源码控制中。相反，你应该忽略生成的CSS文件，在发布时或发布之前用一个可重复的流程来准备样式表，将其作为构建工作中的一步。

但是很多网站并没有也不打算增加这样的步骤，因为他们已经有Sass了。在这些情况下，许多用户把他们编译的CSS嵌入源码控制系统。如果你这样做了，那么合并冲突时有发生。这时最好的建议就是删掉你生成的样式表，在源码控制系统里解决合并冲突（如果存在的话），重新编译。

如果你的应用有了这个构建步骤，它可能足以靠一条命令编译整个项目，但是有些构建系统可能希望通过make这样的工具一次只编译一个文件。在这种情况下，你可以传递单个文件给Compass编译器或Sass命令行编译器，它们都有此类工具的传统界面。

这句命令编译了一个单独的文件：

```
compass compile my_sass_dir/application.scss
```

要用Sass命令行编译一个单独的文件，你需要传递--compass参数：

```
sass --compass my_sass_dir/application.scss my_css_dir/application.css
```

如果你不理解自己团队的构建流程，一定要不耻下问。相比于生成文件存在的合并冲突风险，大多数工程师都喜欢处理一劳永逸的设置。具有复杂构建和发布脚本的团队常常有一个预发环境，在那里他们可以连续部署并进行发布前的最后测试。下面继续介绍两个环境以上的有效管理策略。

7.3.7　和预发服务器一起工作

有些站点有一个预发环境，那里的代码都需要进行最终测试并将要发布上线。有些网站甚至有三层预发环境（边缘、集成、预发），用来测试并集成不同成熟度的功能。

其实有的时候把生产环境的样式表提前放入预发环境就已经足够了。但是预发环境可能会稍有不同——通常是主机名不一样，或者可能你没有使用生产环境的CDN。在这些情况下，你需要调整相应的配置。

针对这种情形，我们有两个建议。一是在编译时设置环境变量：

```
STAGING=true compass compile --force -e production
```

然后在配置文件里，你可以使用Ruby检测这个环境变量并改变相应的设置：

```
if ENV['STAGING']
  relative_urls = true
  output_style = :compact
elsif environment == :production
  relative_urls = false
  output_style = :compact
else #development
  relative_urls = true
  output_style = :expanded
end
```

如果你针对每个环境的配置是明显不同的，那么你也可以创建不同的配置文件：

```
compass compile --forec -c staging_config.rb -e production
```

为了保持一致性并避免重复代码，staging_config.rb文件应该包含一般环境的配置文件，然后在这个基础上进行改动：

```
eval(File.read("#{File.dirname(__FILE__)}/config.rb"))
relative_urls = true
output_style = :compact
```

发布到预发服务器对复杂站点来说是一个很棒的实践，虽然没有原生支持，你还是可以看到作者选择一个Ruby文件进行配置的原因：它允许小概率的特殊情况也能被轻松支持。

现在恭喜你已经把网站放到预发环境了。下一站就是：生产环境！我保证这绝对是一个了不起的成功。

7.4　小结

在这一章，我们看到从早期的快速原型到准备好生产环境的发布，Compass在整个生命周期当中对项目的支持。我们还看到Compass资源辅助器把资源的样式表细节隐藏起来，在开发阶段给你一个简单的配置驱动的本地资源服务方式，在生产环境为网站加速支持多个资源主机。你还学到了一些在浏览器中工作的Compass技巧，以及应该如何让源码控制系统对待你的样式表。在不知道也无需关注资源服务具体知识的情况下，你通过创建智能且高度抽象的样式表为优化网站性能提供了坚实的基础。在下一章，我们会看一看那些互联网上最繁忙的网站们的策略，以及Compass如何支持它们并帮助扩大 。

第8章 高性能式样表

8

本章内容

- 样式表拼接
- 样式表和资源压缩
- 减少和并行图片请求的策略
- 选择器性能和优化策略

通过上一章我们了解到，由于严重依赖于外部资源，样式表的维护相当烦琐。更糟糕的是，样式表还是无数客户端性能问题以及“混合内容”可怕警告的来源。

但是通过遵循客户端性能的最佳实践，你也许可以将页面载入时间缩短几秒。节省的这几秒钟将对你的搜索引擎排名、用户黏度以及转换率目标产生重大影响。关于这个主题的图书数不胜数，本章只介绍Sass和Compass中能让你快速实现Web性能优化的独有特性。

到目前为止，Web性能的一个最佳资源是谷歌的PageSpeed文档：http://developers.google.com/speed/pagespeed/。对于每个关心客户端性能的人来说，该文档都是必读的资料。尽管该文档中提及的一些策略实施起来比较简单，但是很多策略还是颇具挑战性。在CSS中难以“正确做事”时，你很有可能作出妥协并走向错误的方向，而使用Sass和Compass，“正确做事”十分容易。因此，我们希望你能够抽出少量时间来优化你的样式表，尽你所能将Web变得又快又好。

优化样式表基本上可以归结为减少传输的字节数，确保最大限度地将传输数据放入缓存，以及减少浏览器和服务器之间往返的次数。没有什么银弹能保证你的网站变得速度惊人，但是因为Sass和Compass提供的工具和平台可以根据与具体需求相关的使用模式和度量来调整样式表，所以它们已经被许多互联网上的大型站点采用。

但是，如果你一开始并不知道你的网站究竟有多慢，又怎么能够让你的网站变快呢？过去，测量客户端性能是一件琐碎无趣的事，但近几年，测量载入时间和识别页面结构中的关键路径已经容易得多了。

8.1 测量客户端性能

性能优化的起点和终点都是测量。在第一次改变性能之前，你需要知道自己究竟在什么位置。

十多年前只能够粗略地完成这样的测量，而且相关的补救措施也不总是很明显。如今有很多优秀的开发者工具能够帮助你诊断这些指标。如果你过去从来没有使用过下面列出的这些工具，那么你已经错过了页面载入的有关信息，这可是一笔财富。

- ❑ YSlow：http://developer.yahoo.com/yslow/。
- ❑ Google PageSpeed：http://developers.google.com/speed/pagespeed/。
- ❑ WebPagetest：http://www.webpagetest.org/（参见图8-1）。

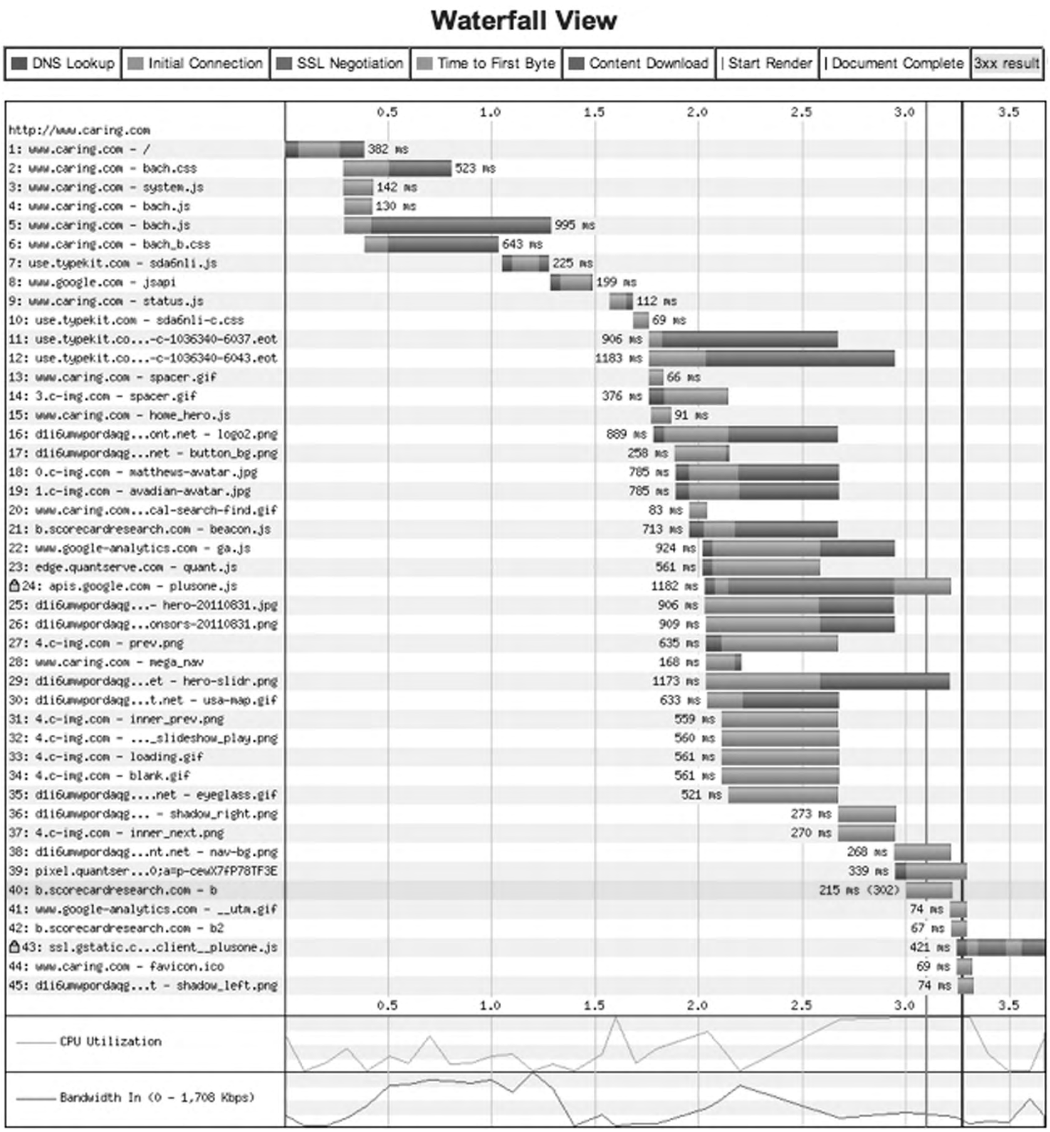


图8-1　webpagetest.org的瀑布图

图8-1所展示的是一张来自webpagetes.org的瀑布图。它是一项能够直接诊断性能问题的免费服务。虽然网站并不漂亮，但是它提供的信息却是惊人的。它能够让你知道究竟是哪些请求阻断了其他请求，哪些请求在并行运行。它能够让你对比缓存与不缓存之间的体验差别，还能够分解DNS所花的时间，接收到第一个字节的时间（服务器到客户端一个来回所需的时间），以及传输时间。

既然你已经了解到了如何测量页面性能，下面就可以开始优化了。优化的第一步是停止导致页面缓慢的不必要工作，最不必要的就是基于CSS的导入。

8.2　回避带有服务器端`@import`的 HTTP 请求

正如你在第3章中看到的，`@import`指令是一个非常有用的工具，它能够将大型的样式表组织为较小的部分，以便轻松地查找样式。在CSS的一个样式表中导入许多其他的样式表并不鲜见：

```
@import url("blog.css");
@import url("forum.css");
@import url("article.css");
@import url("header.css");
@import url("footer.css");
```

这样的写法降低了第一个页面的速度，因为它需要数个HTTP请求以下载所需要的多个样式表。

图8-2的瀑布图展示了浏览器如何处理一张导入其他三个样式表的样式表，被导入的3张样式表同时又各自导入了另外10张样式表。对于一个项目来说，这是完全合理的结构，但是正如你所看到的，只有在CSS文件下载完成后才能开始每一个层级的导入。更糟糕的是，浏览器每次从单个服务器上下载的文件数目是有限制的。直接的影响就是，CSS中的`@import`为第一个页面的载入增加了不必要的时间，而这正是我们最关心的。因此，CSS的最佳实践指出，要将你的样式表连接成少数几个样式表。这就是Sass提供服务器端导入的原因：

```
@import "blog", "forum", "article", "header", "footer";
```

这将提高第一个页面的载入速度，因为所有的样式表只需要一次请求就能下载完成。

但是你是否注意到了其中的缺陷？不同的网站和Web应用有着不同的访问者模式。如果访问者经常返回浏览你的网站，并且样式表中的某些部分几乎不变，那么分开下载一些样式表是有意义的。这能让他们的浏览器在一些样式表发生变化时更好地利用缓存；否则，样式表中的一个小变化就需要再次下载整个网站的样式表。同样，如果你的访问者只浏览网站中的个别页面，那么让浏览器下载每一个模板的样式就没有什么意义了。

许多网站采取的策略是将CSS文件组织为三个层级（参见图8-3）。

- ❑ 核心样式表：几乎每个页面都需要的普通样式。
- ❑ 部分样式表：应用或者网站的某个大型部分需要的普通样式。
- ❑ 单页样式表：只有某个单独页面需要的样式，经常用于销售页面这样设计复杂而独特的页面。

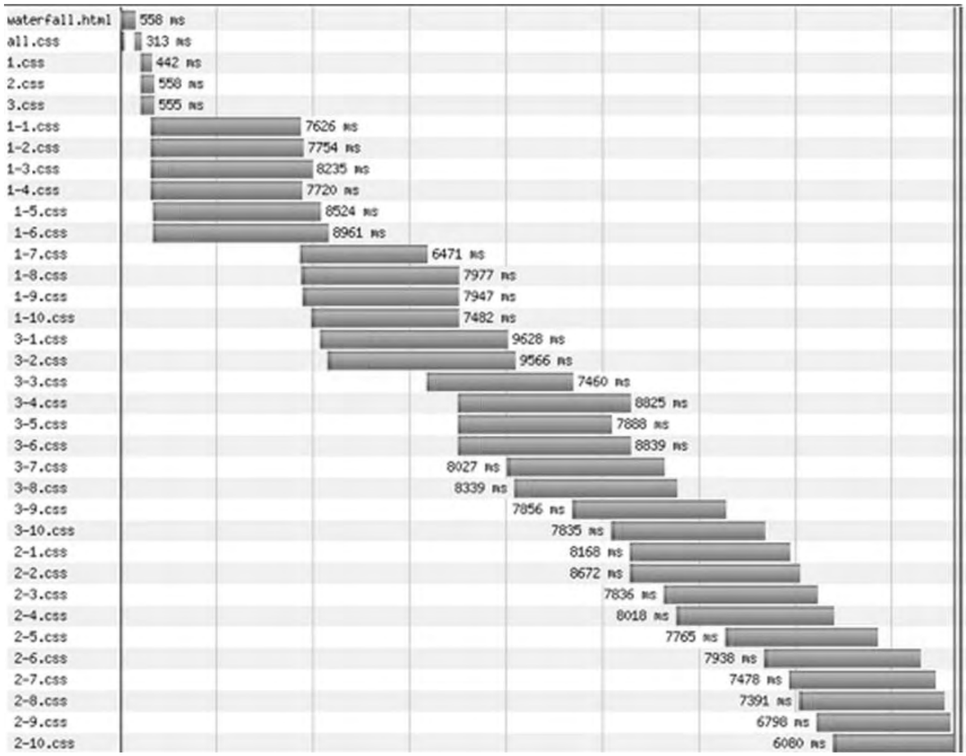


图8-2　基于CSS `@import`的性能瀑布图

这样，既不用牺牲网页组织结构，也减少了多余的请求。接下来，就可以看看怎么减少下载资源（如样式表和图片）的时间。

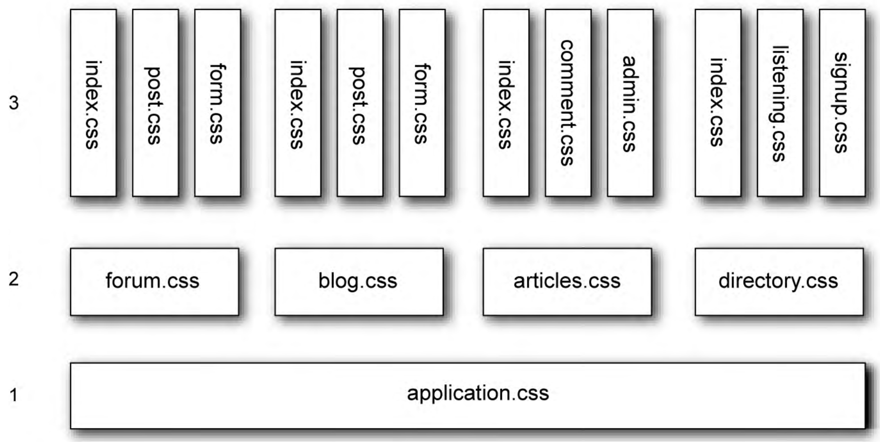


图8-3　常见的CSS结构

8.3　用压缩减少传输时间

有个简单的方法能让你的网站变得更快：缩小在互联网中传送内容的大小。这再明显不过，但是如何做到就不那么显而易见了。

如果你还没有着手，现在是时候开始压缩Sass输出内容了。对大多数用户来说，这是一件非常简单的事情，要做的仅仅是再次连同`-e production --force`参数运行Compass编译器。生产环境默认设置为使用`compressed`输出模式。但是你同样可以通过将Compass配置的`output_style`设置为`:compressed`或者在Sass命令行工具中使用`-t compressed`来完成。

压缩样式表中的文本是一个好的开始。Sass会去除注释和多余的空格，并使用颜色的最简表示法。但这只是个开始，gzip压缩将会帮助你把样式表压缩得更小。

8.3.1　gzip压缩

大多数现代浏览器会连同请求发送一个请求首部`Accept-Encoding: gzip`，只要响应中含有一个包含`Content-Encoding: gzip`的首部，就能被压缩。由于CSS重复、冗长的天性，样式表可以被压缩到原来的10%~15%，确实减少了很多！相比之下，JavaScript文件一般只能被压缩到原有大小的25%左右。

通常，优化前端性能的最佳方式是尽可能缩小资源的网络复写。对于所有关注客户端性能的网站来说，使用gzip压缩文本资源并优化网络图片都是必行之事。

大多数Web服务器都有一个设置选项或者插件来开启实时压缩。在大多数情况下，压缩所花费的时间要远远少于在互联网中传输所节省的时间。

预压缩样式表并根据请求首部提供不同的文件也是一种可行办法。设置方法应该依据服务器来具体分析。但是如果你需要进行这样的设置，Compass提供了一种简单的方式来自动生成压缩样式表。你可以在Compass中注册一个回调函数，每当保存一个新的样式表，它都会运行。将下面的代码添加到你的Compass配置文件中即可：

```
on_stylesheet_save do |filename|
   # run the gzip tool on the file
   # generates a file of the same name
   # plus a .gz at the end.
   `gzip -f #{file}`
end
```

我们鼓励你去研究如何为站点开启gzip压缩。尽管会花费一些时间，但这可以为你的用户提供便利，绝对是值得的。

可惜，样式表仅仅是你需要压缩内容中的一小部分。通常来说，图片才是能做出最大改善的部分。

8.3.2　图片压缩

本节内容与Sass和Compass无关，但是我们觉得应该在此和你分享一些关于图片压缩的基本内容。

关于图片，你首先需要了解的是，大多数图片格式拥有内建的压缩方式。基本的法则就是，针对不同的图片，你应该使用能够达到最大压缩程度的不同图片格式。也就是说，你一般应该使用下面几种格式：

- ❑ GIF格式用于颜色数量少的小文件；
- ❑ JPG格式用于在图片质量设置为最低值时不会引起图片质量明显下降的摄影图片；
- ❑ PNG格式用于其他图片。

PNG是一种能够处理多种图片类型的复杂格式。确保移除其中的alpha层，否则会得到透明效果。我们强烈推荐你安装免费工具Pngcruch，并且在所有的PNG图片上运行一遍。你可以从http://pmt.sourceforge.net/pngcrush/下载Pngcrush。

现在你已经将资源压缩得尽可能小，那么就可以将注意力转移到如何载入这些资源上了。例如，你有没有意识到浏览器会限制从一个Web服务器连续下载资源的数量？如果你运行着一个负载均衡的网站，这对你来说应该有些保守了，因为你肯定可以处理更多的同步连接。你只需要掌握一个叫做*资源托管*的小技巧。

8.4　用资源托管提高页面加载速度

HTTP/1.1标准中指出，Web浏览器应该限制每次页面请求中单个域的同步下载数量，从而达到最佳效果。但是在许多服务器上运行的负载均衡网站可以很好地处理这种突发流量，因此你需要对浏览器耍一点小把戏来解决这一问题。一种普遍的策略是注册多个域（或者子域），并将它们解析到同一个地方。随后，Web浏览器将同步下载更多的资源，使你的用户体验到更快的页面加载速度。

例如，一个叫做example.com的站点可能需要设置img-1.example.com、img-2.example.com、img-3.example.com和img-4.example.com。每一个DNS名实际上都被称为一个`CNAME`记录，意味着它们都是www.example.com的别名。

除了具有并行的好处，设置资源托管来使用一个*无cookie域*也很重要，即一个不会和站点分享cookie的域。它每次可以使用更少的字节向服务器发送图片请求。这并不是什么了不起的事，但是如果你正在努力设置新域，你基本上可以免费获得这个小而重要的帮助手段。关于无cookie域的详细信息，你可以查看谷歌的PageSpeed文档：http://developers.google.com/speed/docs/best-practices/request/。

然后，将指向图片、样式表和JavaScript的链接均匀分发至有效的资源托管。很重要的一点是，总是通过同一个资源托管来获取相同的资源，否则该资源就会被多次下载。很明显，只有在你拥有一个可支配的框架去处理这些繁重的工作，并排除人为引发的错误时，上述的解决办法才是可行的。你非常幸运！Compass使资源托管变得非常容易！

8.4.1　使用资源托管生成URL

在默认情况下，资源托管在Compass项目中是关闭的，但是你可以告诉Compass如何在资源托管上分发资源，从而开启资源托管。例如，为了在四个子域中分发资源，你需要在Compass配置中添加以下Ruby代码：

```
asset_host do |asset|
  host_number = (asset.hash % 4) + 1
  "http://img-#{host_number}.example.com"
end
```

下面进行解释。正如你在前一章中学到的，你应该使用Compass的资源URL辅助器来编写所有的资源引用。如果你已经遵循了这条最佳方式，其实就已经对资源托管有了一些不错的实践。这里，你仍然编写下面的CSS代码：

```
#logo {
  background-image: image-url("logo-small.png")
}
```

你在配置中定义的`asset_host`函数接收一个叫做`asset`的参数，它将被完全解析为指向资源的HTTP绝对路径。根据你的其他配置，解析后的路径格式应该类似`/images/logo-small.png`。`asset_host`函数的任务是返回资源生成URL的协议和主机地址。

尽管你可以自由地运用任何满足需求的逻辑，但是前面的例子在通常情况下已经足够了。首先，`asset.hash`会给你一个数字来唯一地表示`asset`字符串。其次，取模运算符（`%`）会返回其除4以后的余数，该余数是一个0到3之间的整数。最后，它通过每次增加1来进行基于1的计数。在第二行代码中，将`host_name`插入一个字符串来生成合适的返回值。函数中的最后一个值是实际的返回值，这里不需要显式地指明`return`。

接着，Compass会将该资源托管值与被传入的地址和任何缓存相连接，来生成完整的URL：

```
#logo {
   background-image:
      url('http://img-3.example.com/images/logo-small.png?1298578273');
}
```

使用资源托管可以显著地减少客户端渲染时间。资源托管一件非常简单的事情，我们没有理由不采用。

8.4.2　避免内容警告和基于域的资源相混合

如果你的站点支持SSL访问，同时你想要使用资源托管，非常重要的一点是确保你的用户不会从浏览器中接收到不安全资源的警告。处理这个问题的最佳方式是使用相对于协议的URL：

```
#logo {
  background-image:
    url('//assets3.example.com/images/logo-small.png?1298578273');
}
```

你可能会想，http:到哪去了？在这里我们不需要http:。当浏览器遇到一个相对于协议的URL，它会使用进行最初请求时的协议（对样式表的请求）。主要浏览器都支持相对于协议的URL，包括IE6。为了配置Compass在资源托管时使用一个相对于协议的URL，你需要在配置中添加以下代码：

```
asset_host do |asset|
  host_number = (asset.hash % 4) + 1
  "//img-#{host_number}.example.com"
end
```

如果你决定在HTML标记中使用相对于协议的URL，有一点需要注意。IE6和IE7中的一个bug会导致一个指向相对于协议URL的样式表`<link>`标签被下载两次。

IE6和IE7在一个叫做data URI的URL方面也做得很差。但是这并不意味着你不能给使用较好浏览器的用户提供性能的提升。Sass和Compass使你能够轻松地使用内联data URI。

8.5 内联 data URI

CSS3的一个神奇特性是，你不需要图片就能创建出漂亮的效果，但是你依然依赖图片来为设计锦上添花，引起人们对重要部分的关注。在本地进行开发时，你常常会忘了图片并不是即刻就能载入的。虽然在台式机上，低延迟的宽带连接越来越普遍，然而国际浏览用户和移动设备用户常常感觉回到了拨号上网连接的年代：低带宽和高延迟的数据往返时代再次出现了。

想象一下，如果你提供的图片内嵌在样式表中，你就可以避免HTTP数据往返的相应消耗。幸好，我们已经有了一种叫做data URI的机制来实现这种可能性，如图8-4所示。

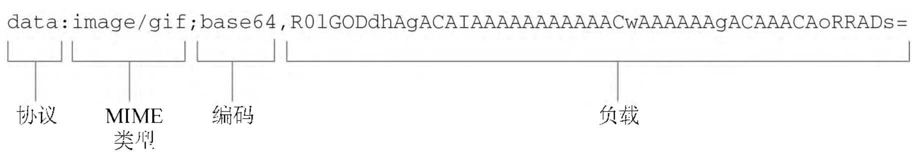


图8-4　data URI的分解

图8-4展示的URI代表一张2×2像素的黑色图片。原图像中的35字节数据已经通过base-64编码方式转化为了70字节的data URI。将它输入到浏览器的URL框中，你会看到一张小图片。

在CSS中使用data URI时，你可能会使用一个Web服务来上传图片并输出相应的data URI，然后对需要内嵌的每个图片重复这一过程，但是这样的做法耗时耗力而且难以维持。通过使用Compass，处理内嵌图片和其他资源简直是小菜一碟：

```
.icon { background: inline-image("black-dot.png"); }
```

Compass可以基于图片的扩展名找出大多数图片格式的MIME类型。但是如果Compass没有识别出图片的扩展名，并且扩展名和MIME类型的第二部分不相同，你可以显式地提供MIME类型：

```
.icon {
  background: inline-image("black-dot.bitmap", "image/bmp");
}
```

既然Compass可以如此简单地处理内嵌图片，并且能够通过避免额外的数据往返带来诸多好处，为什么我们不一直使用这种方法呢？原因有以下几点。

- 体积膨胀——base-64编码算法的效率并不如普通的二进制编码。使用base-64编码时，数据大小会比原来的二进制编码多出大约20%。即使你压缩了样式表，但如果其中包含了大量的内联数据，编码消耗带来的坏处还是远远超出了HTTP延迟带来的坏处。首要法则是，你应该小心任何超过1KB的内嵌数据。
- 缓存——即使这样的做法加快了速度，但是内联图片会增加你的CSS文件体积。一些用户代理也许会选择不把过大的文件放入缓存。例如，iPhone不会缓存大于25KB的文件。另外，改变一张内联图片将引起整个样式表发生改变：如果每张图片都需要一个单独的请求，只有那张发生变化的图片需要被再次请求。
- 浏览器支持——IE6和IE7并不支持data URI，IE8支持的data URI上限为32KB。为了解决这个问题，你可以使用一张备用图片来支持旧版本的浏览器：

```
background-image: inline-url("logo.gif");
*background-image: image-url("logo.gif");
```

如果你这样做，使用旧版本浏览器的用户将会下载这张图片两次，从而付出“性能罚金”。如果你需要支持旧版本的浏览器，可以考虑将data URI放在一个单独的样式表中，并将其设置为选择性连接。

如果内联图片不能很好地满足你的需求，另一个提高图片加载速度的好策略是使用第6章中讨论的精灵图。

到目前为止，我们已经看到了几种通过缩小文件大小和减少HTTP请求次数来改善性能的方法。接下来，我们将处理浏览器渲染，并学习如何优化Sass来提高选择器性能。

8.6　选择器性能

查看一个JavaScript文件如何拖慢页面速度很容易。只需要做一个无限循环，你就能看到浏览器忽然间停止工作了。CSS选择器没有循环，并且比JavaScript要快得多，因此我们很难看到样式表如何影响Web页面的加载速度和总体的响应能力。除去数据传输和解析样式表的时间，选择器的数目和结构能对页面速度产生微小却可以衡量的影响。对于一个大型网站来说，这个影响可能会高达到几百毫秒。当这几百毫秒变成影响你的网站性能的最坏因素时，网站就已经处于一个好状态了。

大多数的性能影响因素都符合帕累托法则：你通过20%的工作得到了80%的加速。优化选择器是在你已经完成了前面简单的部分之后才需要进行的步骤。你花费了几周时间却可能仅仅减少了100 ms。对于某些站点，花费这些努力是有意义的。但是如果你选择去做，你需要考虑进行一些重新设计或者重新组织更大型的模板，来确保所做的努力都是值得的。

首先考虑页面载入次序、服务器响应时间，以及网络传输消耗；之后，才应该考虑选择器方面的优化。

8.6.1 积少成多的问题

选择器速度很快，真的非常快。但是当你拥有上万个选择器时，计算其层叠和继承，以及解析一个文档中元素属性的时间却可以叠加起来，尤其是当文档本身拥有许多元素时。

选择器的性能之战主要集中在两个方面：修剪和匹配。在修剪方面，浏览器迅速确定并排除所有不会匹配某一个特定元素的选择器。在最低限度上，浏览器会查看关键选择器（通常情况下是最右边的选择器成分），来决定这个选择器能否被修剪。最近，浏览器已经开始引入新的修剪启发式方法，比如ID作用域法。

在匹配方面，浏览器会检查所有不能修剪的选择器，来查看整个选择器是否匹配元素的文档上下文。这意味着就后代或者子选择器查询文档的层级，或者查看兄弟元素。

总的来说，优化选择器的一个好方法是，通过尽量减少为了决定选择器是否匹配而必须考虑的HTML元素数目，来确保降低选择器的复杂性。

8.6.2 过分嵌套的危险

Sass很容易使许多选择器慢下来。现在我们要告诉你：所有用Sass编写的、可爱的、易于展示的样式表都在拖慢你的页面。

紧凑易读的Sass文件有时可能变成巨大的CSS文件，因此非常重要的一点是，在编写Sass时要时刻记住最终生成CSS文件的尺寸和复杂性。特别是Sass新手倾向于使用嵌套选择器来复制HTML结构。这种方法有其确定性。最终，你将会陷入一个比内联样式更难维护的情形中，即使最小的标记修改都会破坏你的设计。而且，由于嵌套时的默认连接符是后代连接符（空格），而且关键选择器通常是一个元素选择器，这意味着你编写的样式正在生成最无效率的选择器。

为了帮助你识别异常膨胀的样式表，Compass提供了一个`stats`命令。要使用`stats`命令，必须首先安装一个叫做`css_parser`的Ruby gem：

```
$ gem install css_parser
```

然后运行Compass的`stats`命令：

```
$ compass stats
```

你可以看到关于Sass文件的所有有用信息。下面列出的是一个在Compass网站的样式表中运行`compass stats`命令的输出。（注意，由于格式的问题，一些列没有在此显示。）

```
| ------------------------- | ----- | --------- | ---------- | ------ |
| Filename                  |  Sass |       CSS |        CSS |    CSS |
|                           |  Size | Selectors | Properties |   Size |
| ------------------------- | ----- | --------- | ---------- | ------ |
| home.scss                 |   387 |       510 |       1579 |  41879 |
| ie.scss                   |   114 |         3 |          4 |    306 |
| screen.scss               |   595 |       932 |       2846 |  68575 |
| core/_base.sass           |  2052 |        -- |         -- |     -- |
| core/_clearing.sass       |   382 |        -- |         -- |     -- |
| core/_extensions.scss     |   192 |        -- |         -- |     -- |
| partials/_ads.scss        |   666 |        -- |         -- |     -- |
```

```
| partials/_blog.scss        |    84 |         -- |          -- |      -- |
| partials/_code.scss        |  4100 |         -- |          -- |      -- |
| partials/_example.scss     |   483 |         -- |          -- |      -- |
| partials/_home.scss        |  2508 |         -- |          -- |      -- |
| partials/_install.scss     |   603 |         -- |          -- |      -- |
| partials/_layout.scss      |   683 |         -- |          -- |      -- |
| partials/_main.scss        |  1840 |         -- |          -- |      -- |
| partials/_nav.scss         |  2085 |         -- |          -- |      -- |
| partials/_sidebar.scss     |   550 |         -- |          -- |      -- |
| partials/_theme.scss       |  9667 |         -- |          -- |      -- |
| partials/_typography.scss  |  1824 |         -- |          -- |      -- |
| -------------------------  | ----- | ---------  | ----------  | ------  |
| Total (18 files):          | 28815 |       1445 |        4429 | 110760  |
| -------------------------  | ----- | ---------  | ----------  | -------|
```

如你所见，这条命令使我们可以轻松地看到正在生成的选择器数量以及生成的文件大小。生成的CSS文件几乎比源文件大了四倍。Sass和Compass为你提供的帮助很棒，但是必须牢记，要负责任地发挥它们的威力。

8.7　小结

在Web上工作的一件惊人之处是，对于每个需要优化的部分都有许多优化的方法。不幸的是，如果缺乏一个框架来帮助我们快速地完成工作，对其中许多方法付出的努力将得不偿失。正如你所看到的，Compass拥有很多方法来调整你的网站，以适应数千乃至数百万访问者。

正如你所学到的，作为一位样式表作者，你有很多方式来加快站点运行。优化性能并不仅仅是服务器端的问题，恰恰相反，对于很多站点，你最终会发现客户端性能是最难优化的部分。Sass和Compass能在很多方面帮助你，你不感到高兴吗?

在下一章中，你将学习到高级的Sass技巧，来编写更智能、更强大的样式表。我们将学习如何编写更好、更灵活的混合器，如何操纵颜色并进行高级的数学计算，以及如何编写你自己的Sass函数。

Part 4 第四部分

高级 Sass 和 Compass

在前面的三个部分中，你已经学到了Sass和Compass的基本知识以及如何将其应用到样式表编写工作流程中。在第四部分中，我们将探索一些Sass和Compass的高级特性。第9章将介绍Sass怎样处理数据类型和表达式，你可以添加字符串和值，使用智能的单位转换。我们将带领你认识许多用于处理数值、列表和颜色的强大函数。你可以看到Sass能十分轻松地操纵颜色，让你在样式表中创建合适的动态主题。你也将学到如何编写你自己的Sass函数。最后，我们会介绍如何使用`@for`、`@while`和`@each`指令编写循环，以及如何使用`@if`和`@else`指令编写条件样式。

第10章将在你之前所学内容的基础上创建一个Compass扩展。首先，我们将讨论分享样式表和前端代码的不同方法、它们的缺点，以及Sass和Compass怎样使你简单地分享优秀的可重用样式表。你将跟着我们的进程一步一步地编写一个用于创建漂亮CSS3按钮的高级扩展，并学到如何编写一个具有最基本形式的扩展。在你编写和重构扩展时，我们会讨论编写扩展样式表的设计策略和最佳实践。我们将讨论发布扩展的不同方式，你将学习到如何轻松地创建一个Ruby gem来帮助你分发扩展。最后，简单地看一看如何在Github上分享并管理你的开源项目。

读完本书，你将对如何使用Sass和Compass编写更智能、更具可维护性的样式表有一个透彻的理解。你可以使用一系列强大的新工具和新视角来解决Web设计中的难题。通过开源分享你的想法和发现，你也会了解到如何更简单地参与到设计社区中来。编写无聊样式表的日子已经一去不复返了，等待你的将是令人激动的新挑战。

第 9 章

用Sass编写脚本

9

本章内容

- 操纵Sass理解的CSS值
- 使用Sass的内建函数编写高级的数学和颜色主题
- 定义你自己的函数来避免重复计算
- 使用控制指令来创建高级、可重用的混合器

上一章介绍了如何优化样式表，从而在浏览器中达到最佳性能。在本章中，你将学到如何从可读性和可维护性的角度优化Sass，以及如何突破CSS的限制编写出更加智能的样式表。

你已经知道了Sass给普通CSS带来的诸多好处。变量、嵌套法则和混合器都使得CSS的重复减少了。尤其是混合器，它是一种重构样式表中重复样式和模式的好方法，使样式得以重用。

但是变量和混合器本身只能针对你在样式表中使用的表达模式。有时你想要避免一遍又一遍的宽度计算，或者想要一个在不同的条件下拥有细微样式差别的混合器。为此，你需要Sass中更加高级的脚本特性，本章将对此进行介绍。

高级Sass用法的核心是操纵CSS值（作为属性值出现）的能力。Sass支持你在小时候学到的所有算术：加法、减法、乘法和除法。它甚至能够理解单位，知道`5px+10px =15px`。

Sass同样能够理解其他所有的CSS数据类型，例如颜色、名字（例如`bold`或者`center`）和列表（例如`1px solid black`或者`font, font, font`）。它同时还拥有一个自己的数据类型来表示`true`和`false`，用来决定应该使用哪个样式。

除了数值计算，Sass还主要通过函数来操纵CSS值。除了`rgb`或`hsl`等内建CSS函数，Sass增加了一系列自己的函数来做很多有用的事情。其中的很多函数在复杂脚本的上下文中很有用，但是有一些在单独使用时也很有用，尤其是那些操纵颜色的函数。这些函数功能足够强大，能够用一两个基本色来生成整个主题的色值。

Sass同时还允许在属性以外的地方使用CSS值和Sass变量。使用一种特殊的语法能够将其包括在选择器和属性名中。在需要将选择器名或者属性名作为参数传递给一个混合器时，这种方法很有用。

CSS值最高级的用法是控制结构。控制结构允许你控制样式是否被使用，以及在不使用混合器的条件下为同一个样式生成多个变量。如果曾经使用过JavaScript或者其他编程语言，你一定很

熟悉控制结构。但是如果你没有使用这些编程语言的经验，也不要害怕。虽然这是Sass的高级用法，但其实都很直观易懂。

9.1 使用表达式

表达式是在Sass中操纵CSS值的最普遍方式。什么是表达式？只要能放在属性右边的都可以称为表达式。在普通的CSS中，表达式通常是一个简单的值，比如`bold`或`5px`，或者一个值的列表，比如`1px solid black`。在Sass中，表达式不仅能包含变量，也能包含数学运算符；相关内容我们将在本章中提到。

尽管所有类型的值都能在表达式中使用，它们和数字一起使用时最有用。你可以像做算术一样使用数值表达式，可以使用`+`、`-`、`*`和`/`等符号来进行加减乘除运算。操作符的顺序也和算术中一样：`*`和`/`比`+`和`-`优先运算，括号中的表达式先运算。在下面的样式中，我们使用乘法和减法来创建一个简单的网格系统，如代码清单9-1所示。

代码清单9-1 使用Sass表达式

```
$grid-cells: 20;
$cell-width: 25px;
#main {
  $main-width: $grid-cells * $cell-width;
  $main-padding: 10px;
  width: $main-width;
  padding: $main-padding;
  .sidebar {width: ($main-width - $main-padding*2)/4}
}
```

表达式可以出现在属性或变量值中的任何地方，不需要占满这些值。这对于合并属性来说非常有用，像是`border`或`background`，它们由被多个用空格分隔开的属性值组成。比如下面的例子：

```
border: ($something - $something-else) solid blue
```

由于`$something - $something-else`的值为`5px`，属性就变成了`5px solid blue`。括号并不是必须的，但是添加上括号以后更易于阅读。

既然你已经看到了表达式的运作方式，那就让我们回到CSS和Sass中使用的数据类型，看看它们在Sass的表达式中都能做些什么。

9.2 理解数据类型

CSS属性或Sass变量中的每个值都有一个类型，类型不同，运作方式也不一样。`#abcdef`或`violet`等值代表颜色，用来为文本、背景、边框等着色。`100%`或`5px`之类的值代表数值，用来设置宽度、外边距和内边距。`center`或`auto`之类的值代表许多不同的东西，通常是替换值的一个选择或者是一个特殊值。

Sass能理解所有的这些类型，并根据这种理解对表达式中的值进行不同的操作。操纵不同类型的值有不同的方式，但是所有的类型都在某种程度上支持算术运算符(+、-、*和/)。在本节中，我们将深入分析每种类型以及它们使用运算符的方式。

9.2.1　字符串和名字

字符串是CSS中最常见的数据类型。之所以称其为字符串，是因为它们仅仅是字符连成的串。名字仅仅是一个字符串，例如bold、auto、center等。带有引号的名字亦如此，例如"Helvetica Neue"。前者是*无引号字符串*，后者是*有引号字符串*。

除了引号本身，有引号和无引号字符串之间的主要差别还在于哪些字符可以成为字符串中的一部分。有引号字符串可以包含除"以外的任何字符；而无引号字符串不能以数字和特殊字符开头，也不能包含空格以及一些特殊字符，如*或&①。

Sass还将另外几个特殊的构造看成是字符串。最直观的一个是!important，一般来说，它不能被归为无引号字符串（因为它以!开头），但是Sass还是将它归入无引号字符串的行列。url()的值同样被认为是一个无引号字符串，尽管(和)等特殊字符通常不允许在URL中出现。但是url($variable)不被认为是一个字符串，主要是因为它在url()中包含了$variable的内容。

IE浏览器中特定的滤镜值同样被认为是字符串，因为根据CSS标准（也是Sass解析器参照的标准），它们其实都不合乎语法。因此，progid:DXImageTransform.Microsoft.gradient(startColorstr=#550000FF, endColorstr=#55FFFF00)被认为是一个巨大的无引号字符串。同样，CSS3中的calc()函数目前也被认为是一个字符串。

最常用的字符串操作符是+。当我们将一个字符串加到另外一个值上面，无论该值是否为字符串，这两个值都会连接起来形成一个新字符串（见表9-1）。如果字符串带有引号，结果就带有引号；否则，不带引号。如果两个值都是字符串，其中一个带引号而另一个不带引号，那么结果字符串是否带引号，取决于左边的字符串。

表9-1　在字符串中使用+操作符

表　达　式	结　　果
foo + 1	foo1
"foo" + 1	"foo1"
foo + bar	foobar
"foo" + "bar"	"foobar"
"foo" + bar	"foobar"
foo + "bar"	foobar

大多数其他的操作符都不支持字符串。由于历史原因，-和/实际上也会像+一样将字符串连接起来，但是操作符本身会包含在结果字符串中（见表9-2）。

① 这条规则也有例外的情况，但是这属于CSS部分的知识，超出了本书的范围。

表9-2 操作符-和/在字符串中与+作用相同

表 达 式	结 果
foo - bar	foo-bar
foo / bar	foo/bar

和字符串一样，数值也是CSS和Sass中的常用数据类型。

9.2.2 数值

在Sass以及CSS中，数值包括两部分：实际的数值以及（可选的）单位。常用的单位包括px，em和%。

由于Sass能够理解带有单位的数值，所有的操作符都能和单位一起使用。这里遵循科学运算中的单位处理方法：当对带有单位的数值做乘法和除法时，单位也和数值一起做乘法或者除法。

这意味着5em*4px的结果是20em*px，而99px/1in的结果是99px/in[①]。

单位算术在转换单位时非常有用。例如，如果你设置了$pixels-per-em:16px/1em，就可以通过5em*$pixels-per-em来计算5em是多少像素。得出80px*em/em，两个em可以约分，因此结果是80px。Sass可以自动完成这些处理。

做加法或减法时，结果并不总是有合适的单位。例如，5px+10%就没有什么意义，因为Sass不知道怎么在像素和百分比之间转换[②]。在这些情形下，Sass会抛出错误。但是如果Sass知道如何在单位（例如in和cm）之间进行转换，它会自动帮你转换。

大多数数值运算符的作用与我们在学校中学到的相同。除此之外，还有一个（有点复杂的）运算符：取模。取模写作%，它能计算出两个数值相除之后的余数。因此$num1% $num2的结果是$num1/$num2的余数。这个运算符用处不大。

当我们在数值计算中使用/操作符时，会出现一点复杂的情况。CSS允许特定值被一个正斜杠（/）分开，在这种情形中，正斜杠并不代表除号。这在实际情况中很少会用到，但是既然Sass表达式是CSS的一个超集，它需要支持这种语法，而不是将两个数字相除。

Sass处理这种情况的方式是使用几个简单的规则来决定是使用除法还是使用一个普通的/（正斜杠）。如果两值中有任意一个是字符串，结果将使用一个普通的正斜杠。否则，只要以下满足以下三个条件中的任意一个，Sass都将进行除法运算：

- 在/的任意一边使用一个变量；
- 整个值被小括号包围；
- 该值被用作其他算术表达式的一部分。

例如，在下面的表达式中，/不表示除法：

```
1px/2px => 1px/2px
```

但是在下面三个例子中，/表示除法：

① 注意，px*em和px/in是不合乎Sass语法的，为了获得这些值，你必须使用数值进行乘或除运算。

② Sass也支持CSS3中的calc()函数，这个函数能完成这种计算，但这是因为浏览器了解有关布局的所有信息。

```
$var: 1px; $var/2px => 0.5px
(1px/2px) => 0.5px
1 + (1px/2px) => 1.5px
```

尽管用到数值的机会更多，但颜色也是Sass最有趣的数据类型之一。

9.2.3　颜色

CSS中的颜色可以写成许多不同的形式。最常用的是使用RGB通道的十六进制表示法：`#abcdef`表示171红色，205绿色，以及239蓝色。你也可以使用一个函数来表示同样的信息：`rgb(171,205,239)`。另外，也有一些经过命名的颜色，例如`blue`和`violet`，它们更具有描述性，但是可选择的种类比较少。`hsl()`函数和`rgb()`函数相似，但前者拥有更多有用的特性。最后，`rgba()`和`hsla()`分别与`rgb()`和`hsl()`类似，但是它们都允许在指定颜色时设置一个alpha透明度值。Sass能理解所有这些形式。

不论最初的颜色值以何种形式表示，Sass内部会同时使用RGB和HSL来表示颜色。这对于我们将在9.3.2节中谈到的颜色操纵函数很有帮助，它们中的许多都对颜色的HSL属性进行操作。

过去，颜色支持使用`+`、`-`、`*`和`/`操作符以及数字和其他颜色进行运算。但是这些操作符既不直观也不是很有用，因此在9.3.2节讨论的颜色函数中，这些操作符都不再使用了。

我们下面要讨论的数据类型是列表。尽管列表与颜色都经常在CSS和Sass中出现，但你可能还不习惯将列表作为整体来看待。列表在清除重复样式方面大有作为。

9.2.4　列表

列表是一个数值的序列，用来表示`border`或者`background`等属性。例如，`1px solid black`是包含三个数值的列表，你可以将它用于`border`属性。列表中的值用空格隔开，正如这个例子所示；或者用逗号隔开，例如`font, font, font`。

准确地说，Sass中的列表必须包含至少一个项目。但是单个的值也被视为包含具有列表所有性质的单个项目，这些值包括列表函数和`@each`这样的控制指令，本章后面将涉及这些内容。

尽管CSS的列表只能包含个别的值，但是Sass中的列表可以包含其他列表。其中最清楚的表述方式就是在用逗号隔开的列表中加入用空格隔开的列表。例如，列表`foo bar, baz bang, bip bap`包含三个元素：`foo bar`、`baz bang`和`bip bap`。其中每一个元素也是列表，分别包含两个单词。

列表中也可以使用括号来嵌套同类型的其他列表，比如`(foo bar) (baz bang) (bip bop)`和`(foo, bar), (baz, bang), (bip, bop)`，它们都是由三个包含两个元素的列表组成的列表。当包含列表的列表被转换成CSS时，括号会被移除，以保证结果是有效的CSS。

算术操作符对列表来说没什么用，尽管可以在列表中使用，但也只能将列表简单地转换为无引号字符串，此时会使用相应的字符串操作符。

列表真正的有用之处体现在两个方面。第一，它使用`@each`指令使代码更加简练，该指令的讲解参见9.5.2节。第二，可以用它将更加复杂的参数传递给混合器，然后这些参数就能被Sass函

数所用，9.3.3节将涉及这一点。

Sass中的最后一种数据类型并非来源于CSS，但是引入它可以让混合器使用逻辑和做出选择。

9.2.5 布尔值

布尔值以逻辑学家乔治·布尔命名，用来代表真值。布尔值只有两种：`true`和`false`。它们在Sass中（连同`@if`指令；我们将在9.5.3节中讲述`@if`指令）用来决定应该使用哪种样式。

布尔值不使用算术操作符，它们有自己的操作符：`and`、`or`和`not`。这些操作符都很直观：如果`$bool1`和`$bool2`都为`true`，那么`$bool1 and $bool2`为`true`；如果`$bool1`或者`$bool2`其中一个为`true`，那么`$bool1 or $bool2`为`true`。`not`操作符只在一个值上进行操作：如果`$bool`为`false`，那么`not $bool`为`true`；反之，则`not $bool`为`false`。

事实上，`and`、`or`以及`not`操作符可以和任何类型的值一起使用，尽管它们和布尔值一起使用时最有用。对于`and`、`or`和`not`操作符来说，数字、颜色、字符串和列表都被看做是`true`。在使用非布尔值时，操作符`and`和`or`返回的结果也是非布尔值：除非`$val1`为`false`，则`$val1 and $val2`将返回`$val2`；只有`$val1`为`false`，`$val1 or $val2`才会返回`$val2`。

有一些作用在其他类型上的操作符也返回布尔值，如表9-3所示。

表9-3 返回布尔值的操作符

操 作 符	含 义
<	小于
<=	小于或等于
>	大于
>=	大于或等于
==	完全相等

小于和大于操作符只作用于数字，但是==操作符可以作用于全部类型[①]。这些操作符都返回布尔值。

对于许多数据类型来说，操作符并不足以让用户充分利用这些类型。鉴于此，Sass通过函数来弥补这一点，并展现出了更多的脚本功能。

9.3 函数

Sass中的函数可以对一个或多个值（变量）进行特殊操作。它们使用的语法和诸如`rgb()`或者`hsl()`的CSS函数类似，但是它们被看做Sass表达式的一部分，而且返回的是Sass值。

和CSS函数不同，Sass函数可以使用关键字变量。这意味着我们并不是通过参数的顺序来指

① 回忆一下，Sass能够理解所有颜色的RGB和HSL值，因此`blue`、`#0000ff`和`hsl(240,100,50)`在==操作符看来都是同一种颜色。

明参数，而是可以显式地命名几个或者所有的变量。该特性的语法是$name: value，参数的名字和函数一起列出。这对于拥有许多参数的函数来说尤为有用，你不需要记住哪个参数应该放在什么位置：

```
rgb($green: 127, $blue: 127, $red: 255)
```

Sass设计哲学的一个重要方面是：不可能总是了解所有浏览器支持特性的最新状况，你最好也不要尝试去了解。因此，如果你使用了一个Sass无法识别的函数，Sass会认为它是一个普通的CSS函数并不加转换地进行传递（参数除外）。所以，在输入函数名时打错字将是一件非常烦人的事情，一定要牢牢记住这一点。

大多数的Sass内建函数都被设计为可以在不同的场景中广泛使用，但并不能保证能够处理所有数值或颜色的操作。因此，Sass允许用户定义自己的函数，它的定义方式和定义混合器的方式类似。用户自定义函数将在本节的结尾讲到。

大量Sass函数被设计出来帮助你轻松自定义函数和混合器。许多Sass的数字函数在用户日常的设计工作中并不是很有用，但是当你在编写一个找出网格宽度的函数时可以从中获得一些帮助。类似地，大量的函数可以用来提供关于值的信息，以便函数和混合器基于这些信息做出决定。

在本节，我们将介绍大部分常用的Sass内建函数，然后讨论如何编写你自己的函数。我们将从数值函数开始讲解，其次是颜色函数和列表函数，最后概述无法归入特定类别的函数。

9.3.1　数值函数

Sass中有一些函数可以使处理数值的工作变得轻松，尤其是在需要频繁使用Sass编写样式表的时候。目前，Sass中并没有与复杂数学操作相关的函数，例如指数函数、对数函数和三角函数，因为这些复杂的数学运算在样式表中几乎没有什么用。如果真的需要使用这些运算，可以通过Ruby扩展API轻松地添加它们。表9-4列出了我们将要讨论的函数。

表9-4　用于数值的函数

函　　数	描　　述
`abs($number)`	取$number的绝对值
`ceil($number)`	$number向上取整
`comparable($number-1, $number-2)`	返回$number-1和$number-2是否能比较
`floor($number)`	$number向下取整
`percentage($number)`	将小数$number转换为百分数
`round($number)`	将$number转换为最接近的整数
`unit($number)`	返回$number的单位
`unitless($number)`	返回$number是否没单位

其中的三个Sass数值函数将约数转化为整数。不像其他的一些编程语言，当你在Sass中将两个数相除时，结果通常是一个小数而不是一个整数。通常情况下，这就是你想要的结果，但是有时你想要一个整数。约数函数就是用来处理这种情况的。ceil($number)（取上整数）向上取

整数，floor($number)向下取整数，round($number)四舍五入取最近的整数。

和约数函数相似的是abs($number)，它用来返回一个数字的绝对值。如果$number是正数，abs返回这个正数本身；如果$number是负数，abs返回$number的相反数。

percentage($number)是另一种类型的函数。它接收一个小数（例如0.6，0.33，或者1.3），然后返回一个百分数。因此0.6变为60%，0.33变为33%，1.3变为130%。它的作用和$number*100%相同，但是阅读起来更容易。

Sass中也有一些数值函数用来提供关于一个或多个数值的信息。unit($number)将一个数值的单位作为一个字符串返回，当一个混合器的参数单位发生错误需要输出错误信息时，这相当有用。如果数值没有单位，则unitless($number)返回true，否则返回false。comparable($number-1, $number-2)基于两个数字的单位，返回两个数字能否相加和比较。例如，comparable(13in, 5cm)返回true，因为英寸和厘米可以相互转换；但是comparable(5px, 10%)返回false，因为像素和百分率不能用同一标准来衡量。

总体上来说，Sass中的数值函数处理的都是普通的数学操作，而Sass处理颜色的函数则更加丰富有趣。

9.3.2 颜色函数

Sass中的颜色函数可以大体分为两类：返回颜色信息的函数，将旧颜色转换为新颜色的函数。表9-5列出了所有的颜色函数，并将它们归类到了信息和转换两大类中：

表9-5 用于颜色的函数

函 数	类 型	描 述
alpha($color)/opacity($color)	Informative	返回$color的alpha通道
blue($color)	Informative	返回$color的蓝色通道
green($color)	Informative	返回$color的绿色通道
hue($color)	Informative	返回$color的色度属性
lightness($color)	Informative	返回$color的亮度属性
red($color)	Informative	返回$color的红色通道
saturation($color)	Informative	返回$color的饱和度属性
adjust($color, ...)	Transformative	按照给定的颜色成分值调整$color的各个属性
complement($color)	Transformative	返回$color的色环与$color的互补
grayscale($color)	Transformative	返回$color的灰度版本
invert($color)	Transformative	返回$color的反相版本
mix($color-1, $color-2, [$weight])	Transformative	按照$weight的百分比将$color-1和$color-2混合在一起，返回混合后的颜色
scale($color, ...)	Transformative	按照百分比调整$color的各个属性
set($color, ...)	Transformative	将$color的各个属性设置为固定值

你在9.2.3节中了解到，无论创建方式如何，所有的Sass颜色都能理解其RBG和HSL值。信息函数让你可以直接获取这些值的单独成分：红、绿、蓝、色度、饱和度和亮度，以及alpha通道（通常是1，表示一个完全不透明色，除非该颜色由`rgba()`、`hsla()`或者特定的Sass函数创建）。这些函数以其返回的成分名称来命名：`red($color)`、`green($color)`、`blue($color)`、`hue($color)`、`saturation($color)`、`lightness($color)`和`alpha($color)`（也可以写成`opacity($color)`）。

以上每个函数都返回一个特定成分的值，返回值的形式与传递给`rgb()`和`hsl()`函数的参数值形式相同。因此，`red()`、`green()`和`blue()`返回位于0到255之间的数，`hue()`返回一个位于`0deg`到`359deg`[①]之间的数，`saturation()`和`lightness()`返回位于0%到100%之间的数，而`alpha()`返回一个位于0.0到1.0之间的数。

转换函数比信息函数运用的范围更广，因为它们可以快速而简单地生成漂亮的颜色主题。最有用的两个函数是`adjust($color)`和`scale($color)`。这两个函数都接受一个颜色作为第一个参数，后面的参数是一列描述如何转换该颜色特定成分的关键字。这两个函数对每种成分取一个关键字参数（例如`$red`、`$saturation`、`$alpha`），但是它们处理参数的方式不同。

`adjust($color, ...)`函数通过给定的数值，增加或者减少一个或多个成分的值。因此，`adjust($color, $red: 20, $alpha: -0.5)`将`$color`的红色成分增加20，将不透明度减少0.5。同样地，`adjust($color, $lightness: 15%, $hue: 10deg)`将亮度增加15%（新亮度为旧亮度加15%），并将色度增加10度[②]。

`scale($color, ...)`函数和`adjust()`函数类似，但是它接受所有成分的百分率作为参数。不像`adjust()`一样通过一个固定的数值来增加或减少成分，`scale()`通过给定的百分数来流畅地设定这些成分的值。因此，不管原来的亮度是多少，`scale($color,$lightness: 30%)`将使得`$color`的亮度增加30%(更接近纯白30%)。如果初始的亮度数值较高，`scale()`比`adjust()`要更好：例如一个颜色已经有了80%的亮度，`adjust()`会使得该颜色变为纯白（亮度为100%），而`scale()`只会使亮度变为86%。和`adjust()`一样，只要关键字不同时属于RGB和HSL成分，`scale()`可以接受任意多个关键字。`scale()`不支持`$hue`，因为色环是一个圆，衡量它没有什么意义。

另一个有用的颜色函数是`mix($color-1, $color-2, [$weight])`[③]。这个函数通过计算各个成分的平均值来将两种颜色混合到一起。另外，你还可以有选择地通过`$weight`参数来决定两种颜色对混合颜色的分别影响。`$weight`越接近100%，使用的`$color-1`越多；`$weight`越接0%，使用的`$color-2`越多。这个混合色还受两个颜色的透明度影响，透明度越小的颜色对结果颜色的影响越大。

---

① CSS标准指出HSL颜色中`hue`成分的单位是`deg`。这个单位是可选的，因此在通常情况下会被省略。

② 你在使用RBG关键字时不能使用HSL关键字；除此限制之外，你可以使用任意多的关键字。

③ `$weight`周围的方括号表示这是一个可选参数。

set($color)函数与adjust()和scale()函数类似：它对颜色的每种成分都接受一个关键字。但是它的行为更加简单（因此有用程度下降）：它将这些成分设定为一个新值，而不是对原有成分的值进行修改。因此set($color, $red: 120)会将$color的红色成分设置为120，仅此而已。

最后，我们还要提及一些非常方便的颜色函数。如果没有这些函数，我们也可以实现这些函数的功能，但是这些函数使得这些功能变得更为简单明了。grayscale($scale)将$color的饱和度设置为0%，从而将颜色变为灰度。complement($color)将色度旋转180度，从而使得色轮与原来的颜色互补。invert($color)翻转变换所有的RGB成分，从而返回原色的反相版本。

Sass提供了三个内建的列表函数，我们会在接下来一一讲解。

9.3.3 列表函数

通过使用Sass的列表函数，你可以对列表进行任何需要的操作。而对于一些重复的操作，更好的方法是定义你自己的函数。

最有用的列表函数是nth($list, $n)，它会返回列表中的一个单独项目（第*n*个）。不像JavaScript等编程语言，Sass列表中的项目从1开始计数。因此，nth(foo bar baz, 2)返回bar，nth(a b c, 1)返回a。

join($list1, $list2, [$separator])函数用来创建新的列表。它将两个列表连接起来。由于单个值被视为只有一个项目的列表，这个函数也可以用来从单个项目创建列表。可选择的$separator参数用来决定列表的类型，它可以是空格或者逗号。如果没有指明$separator，那么合并后的列表类型取决于$list1的类型。

length($list)函数很简单。它返回$list列表中项目的个数。因此length(1 2 3)为3，length(foo)为1。

Sass中同样有很多无法归类的杂项函数，它们对于编写高级的脚本非常有用。

9.3.4 其他Sass函数

Sass的杂项函数大多用来编写在很多项目中使用的混合器。type-of($value)函数返回一个无引号字符串，代表问题中值的类型。这个类型可以是number、string、color、bool（对于布尔值来说）或者list。

if($condition, $if-true, $if-false)根据一个布尔值在两个值之间进行选择。如果$condition为true，那么返回$if-true，否则返回$if-false。和布尔操作符一样，对于if()来说，任何的非布尔值都被视为true。

你也可以通过使用@function指令定义自己的Sass函数。

9.3.5 用户自定义函数

当你在很多上下文中需要进行一些重复的数值计算和颜色转换时，用来自定义函数的

@function指令非常有用。@function指令和@mixin非常类似，不同之处在于@function包含的内容较少，每个@function必须返回一个结果：

```
@function grid-width($cells) {
  @return ($cell-width + $cell-padding) * $cells;
}
```

@return指令是@function的核心。它和JavaScript中的return很像：接收一个Sass表达式，然后将表达式的值作为函数的结果返回。它也会立刻结束函数，这在你了解控制指令之后会十分有用。

在一个函数内部添加一条CSS规则并没有什么意义，添加一个属性也没有什么意义。事实上。一个@function仅仅包含几样内容：首先当然是@return，还有一些变量声明、注释以及控制指令（我们将在9.5节讲述）。

9.4　在选择器和属性名中使用表达式

CSS属性并不是Sass中唯一可以使用CSS值的地方。为了使用CSS值、变量和表达式，Sass在许多额外的上下文（如选择器和属性名）中增加了一项特殊的语法。这对于混合器来说非常有用，尤其是那些被广泛使用的混合器，因此在编写混合器时需要尽可能地考虑其可适用性。

你可以在选择器和属性名的任何地方将一个表达式包裹在#{和}之间，表达式的结果将会在CSS输出结果中代替#{...}。这个结果将会像属性值一样出现，不过有引号字符串的引号会被移除。这种做法叫插值。

下面的例子利用插值创建一个混合器，并在选择器和属性名中使用其参数，如代码清单9-2所示。

代码清单9-2　使用最终值替换表达式

```
@mixin thing($class, $prop) {
  .thing.#{$class} {
    prop-#{$prop}: val;
  }
}

@include thing(foo, bar);
```

#{$class}被foo所替换，#{$prop}被bar所替换，因此最终的CSS如下所示：

```
.thing.foo {
  prop-bar: val;
}
```

除了打包一些样式之外，插值还能使混合器完成更多工作。例如，当我们使用一些由浏览器提供的最新CSS特性时，混合器可以用来移除生产商前缀的重复编写。在下面的例子中，experimental混合器使用插值来避免为最新CSS性质逐一编写浏览器前缀，如代码清单9-3所示。

代码清单9-3 使用插值来添加CSS属性的生产商前缀

```
@mixin experimental($property, $value) {
  -moz-#{$property}: $value;
  -webkit-#{$property}: $value;
  -ms-#{$property}: $value;
  #{$property}: $value;
}
```

尽管对于特定的浏览器CSS hack非常令人讨厌，然而有时候它们也是很必要的。插值也会很有用。在下面的例子中，当给定一个针对于IE6的值时，我们使用插值来避免两次编写属性名，如代码清单9-4所示。

代码清单9-4 使用插值来编写CSS浏览器hack

```
@mixin bang-hack($property, $value, $ie6-value) {
  #{$property}: $value !important;
  #{$property}: $ie6-value;
}
```

插值也可以用在表达式中，尽管由于表达式本身已经允许变量和其他表达方式的使用，插值在其中的用处并不是很大。但是它确实能够将变量插入到字符串以及类字符串的表达式（像`calc()`或者IE浏览器的`filter`语法）中，如代码清单9-5所示。

代码清单9-5 在字符串中插入变量

```
content: "This element is #{$color}";
width: calc(10% + #{$padding});
filter: progid:DXImageTransform.Microsoft.Alpha(
              Opacity=#{$opacity * 100}
         );
```

可以看到，插值在编写动态样式表时非常有用。它可能并不是你每天都会用到的技巧，但绝对是你需要了解掌握的一个好技巧。

9.5 控制指令

从某些方面来说，控制指令是Sass脚本编写中的最高级技巧。Sass中的大部分内容都基于CSS，而控制指令让Sass更像是一门编程语言。但即使你是一个完全没有编程经验的设计师，也不要被“更像编程语言”这样的说法吓到：所有的Sass控制指令都非常直观。在进行设计和编写对设计有帮助的混合器（这是必须的，否则它们不会成为Sass的一部分）方面，它们非常有用。

控制指令是一种特殊类型的指令，它控制Sass样式块变为CSS的方式。控制指令遵循`@directive{...}`的形式，控制位于Sass块中的样式。每种指令的运行方式各有不同。我们将在本节中查看三种Sass的控制指令。前两种是`@for`和`@each`，它们可以多次使用CSS样式块，每次都略有变化；`@for`的运行次数由指定的数字决定，而`@each`则会迭代列表中的每一个项目。最后一种是`@if`，它控制CSS样式是否得到了使用。

控制指令并不是仅仅用来编写样式。在编写复杂的混合器或者函数时，在控制指令中仅包含变量赋值可能会很有帮助。控制指令允许你使用`@if`来选择变量的定义，或者使用`@for`或者`@each`来增强一个值。

我们首先来介绍`@for`指令。

9.5.1　对数字重复样式

`@for`指令用来计数，每数一个数字则使用一个样式块。它有两种相似的语法：`@for $i from 1 to 5 {...}`以及`@for $i from 1 through 5 {...}`。对于两种语法，首先设置变量`$i`为`1`，然后每使用一次块中的样式就增加1。两种语法的不同之处在于`$i`停止计数的位置：对于`1 to 5`，它停在4；而对于`1 througn 5`，它停在5。

下面的样式会被编译为5个不同的等级类，每一个类都有一张不同的背景图片。假定这些图片包括1到5个星形、大拇指等：

```
@for $i from 1 through 5 {
  .rating-#{$i} {
    background-image: url(/images/rating-#{$i}.png);
  }
}
```

你不一定要使用1和5作为开始和结束的数字。你也可以使用-5和15或者22和379。你甚至可以使用变量，从`$a`数到`$b`（这对于混合器和用户自定义函数非常有用）。同样地，你可以依照自己的喜好对计数变量（本例中为`$i`）命名。

`@for`并不能解决所有的问题：它不能向下计数，不能隔位计数，也不能对分数计数。但是运用一点小技巧，你可以通过对`$i`进行一些数学计算来模仿这些计数方式，如代码清单9-6所示。

代码清单9-6　模仿向下计数或隔位计数

```
// 从10到0向下计数
@for $i from 0 through 10 {
  $i: 10 - $i;
  ...
}

// 从0到20隔位计数
@for $i from 0 through 10 {
  $i: $i * 2;
  ...
}
```

尽管计数很有用，但是你经常只需要针对一个列表中的每一个值编写样式。针对这种情况，你可以使用`@each`指令。

9.5.2　对列表重复样式

像`@for`指令一样，`@each`指令多次重复一个样式块。但是`@each`不仅仅是计数，而是对一个

列表中的每一个项目使用样式块。它的语法是`@each $item in foo, bar, baz {...}`，会依次将`foo`、`bar`和`baz`的值赋给`$item`变量。下面的例子就是使用`@each`指令对一个网站的每个部分编写链接的样式：

```
@each $section in home, about, archive, projects {
  nav .#{$section} {
    background-image: url(/images/nav/#{$section}.png);
  }
}
```

`@each`指令中的列表可以用空格分开，也可以用逗号分开；用逗号分开的列表更易于阅读。它也可以是一个变量的列表，甚至可以是一个包含其他类型值的变量（因为非列表值被视为单元素列表）。

9.5.3 条件样式

`@if`指令用来控制一个样式块是否得到使用。它在样式表日常编写工作中的使用次数可能没有`@for`或者`@each`多，但是它在编写跨项目使用的混合器和函数时出现的频率很高。这些混合器和函数需要接收数量非常多的参数，有时需要基于这些参数表现出不同的行为。`@if`为这些需求提供了解决办法。

`@if`指令的语法是`@if condition{...}`，其中的`condition`可以是一个布尔值，一个带有布尔值的表达式（例如使用==或者>的表达式），甚至一个带有其他值的表达式（因为这些值被视为`true`）。如果表达式为`true`，那么这个样式块就在使用中，否则，这个样式块就被忽略了。下面的例子使用`@if`指令来为样式添加浏览器前缀，前提是已经设置了`$use-browser-prefixes`变量，如代码清单9-7所示。

代码清单9-7 使用`@if`的条件样式

```
.rounded {
  @if $use-browser-prefixes {
    -moz-border-radius: 5px;
    -webkit-border-radius: 5px;
  }
  border-radius: 5px;
}
```

你也可以有选择地在`@if`块之后包含`@else`指令。这样你就可以在没有使用第一个样式块时使用另一个样式块。`@else`指令可以拥有自己的条件，语法为`@else if condition {...}`，只有在条件为`true`时才会使用该样式块；它也可以不带条件，在这种情况下该样式块总是会被使用。每个`@if`指令可以配合使用任意数量的`@else if`指令，但是只可以配合使用一个`@else`指令。下面的例子使用了一个`@else if`和一个`@else`来基于alpha通道值调整背景色，如代码清单9-8所示。

代码清单9-8　将`@if`和`@else`合并起来编写高级样式

```
@if $alpha < 0.2 {
  background-color: black;
} @else if $alpha < 0.5 {
  background-color: gray;
} @else {
  background-color: white;
}
```

上面的例子会在接收`$alpha`作为参数的混合器中出现，但可能并非体现为任意非混合器样式。

9.6　小结

现在你已经了解了Sass的所有功能。适当地使用这些工具，可以创建强大的、可重用的样式表，它们可以大大地减轻设计时的工作量，增加设计的表达性。你甚至可以编写出你和他人都可以使用的跨项目样式表，以此改善广阔的设计界。

在本章中，你学到了Sass表达式是如何运行的，包括如何在表达式中使用算术操作符和变量。我们还讲述了Sass支持的数据类型及其操作符。

我们接着介绍了许多由Sass提供，用来处理这些数据类型的有用函数，以及如何编写自定义函数。你学到了通过插值在选择器和属性中使用这些函数和表达式的结果。最后，我们谈论了如何使用控制指令来控制样式块在编译的CSS中的使用方法和使用频率。

到目前为止，你已经在本书中学到了许多关于Sass和Compass的知识，看到了Sass作为一门富有表现性的动态语言所展现的威力，以及Compass如何整合你的开发和生产环境并提供一个不可思议的Sass工具库。在下一章中，我们将会把前面所学的内容整合起来，学习如何使用Compass扩展来创建一个现代的样式表框架。

第 10 章

创建并分享一个Compass扩展

本章内容

- ❑ 分享Sass样式表以及需要Compass扩展的原因
- ❑ 介绍简单的Compass扩展
- ❑ 详细介绍如何编写一个高级扩展
- ❑ 创建模板来引导或演示一个扩展
- ❑ 简要介绍分享Compass扩展的不同方法

到目前为止，每章的内容都集中在帮助你掌握Sass和Compass的不同特性，以及它们如何通过样式表来提升工作效率。本章将把前面的所有内容整合起来，基于你所学到的Sass和Compass知识来帮助你迈出下一步，并且建立一个Compass框架。我们将探讨在社区中分享代码的相关问题，并详尽介绍如何编写一个用于为按钮添加CSS3样式的扩展。我们将探索创建这个扩展所需的设计方案，并探讨创造真正优秀代码的原则以及最佳方式。

10.1 分享和重用样式表

如果你有过编写样式表的经历，那么一定能够体会看到一些巧妙的CSS代码最终汇集起来时的那种愉悦。我们事先保存样式表并刷新浏览器，然后我们创造的东西就完美地出现在了眼前。成功了！你可能开始准备跳起胜利的舞蹈了。但是下一步应该做些什么？

Sass善于表达的特性可以让我们的样式表更加智能。和使用原生CSS相比，我们需要花更多的时间来探索和发现新的样式表。同时，Sass和Compass也使得我们比过去更加容易地分享胜利和顿悟。

10.1.1 Sass比CSS更容易分享

在过去，分享样式表意味着撰写一篇带有CSS代码片段和可供下载代码示例的博客文章。这种方式的结果往往是，需要解决的问题越复杂，博客的读者就越难正确地使用。有心的作者可能会在文章中解释如何适当地自定义并重用样式表，这意味着那些更加复杂有趣的样式表难以得到分享，难以为读者所使用。另外，如果使用的是CSS框架，任何包含在其中的样式表都会自动影

响网站的样式。这意味着用户必须非常谨慎地使用类名，或者必须编辑CSS框架来选择并挑出想要的部分。

另一方面，Sass的用户可以分享非常有用的样式表，这些样式表不需要输出CSS代码。一个充满着混入类和函数的样式表不会直接对一个网站的样式产生影响。相反，Sass提供了许多工具来帮助设计师完全自主地进行设计。Sass样式表并不局限于选择器、属性以及属性值。拥有变量、`@if`、`@else`、`@for`、`@while`以及自定义函数和混入类，Sass在继承了CSS的基础上具备更强的表现力。

在继承之外，CSS框架的一个致命缺陷在于没有重用的概念。当一个Sass作者在编写`@mixin button-style($color)`时，这个混入类定义了一个具有特定目的的代码块，每当代码中含有这个混入类，它都向读者说明这个元素属于一个有组织的设计框架。而原生的CSS则无法做到这点。CSS代码块无法被重用，只能被复制。在一系列样式表中使用时，CSS代码无法维持它的完整性。

表现力和抽象性是使得Sass代码易于解释与分享的两个关键特征。

10.1.2　分享Sass

如果你之前使用过CSS框架，那么一定熟悉在开始使用框架时必须掌握新类名和设计模式的痛苦。尽管有些框架有更好的文档，但是总的来说，如果你不想要框架提供的东西，就需要费劲心思花数小时的时间去钻研框架的代码来移除你不喜欢的部分，同时试图保留喜欢的部分。这实在是太糟糕了。在分享Sass样式表时，要注意避免这样痛苦过时的实践方式。

尽管你可以像过去的CSS框架一样将最喜欢的按钮、列表、表格以及排版设计打包起来，但是Sass将会引导你走向另外一种分享方式。使用Sass，你无须编写一种特定类型的按钮样式，而是将样式封装到一个混入类内部。将你的样式打包成一系列混入类是分享样式表框架的好办法，可以帮助用户保持对其网站设计的全权控制。另外，你也可以编写接收一系列变量的混入类，这样可以让用户在使用你的框架时自定义属性。

Sass函数也是一个非常好的资源。如果你的框架提供可自定义的配色方案，保证最终结果的质量将是一大挑战。例如，如果用户可以为一个按钮选择一种背景色，按钮上的文本仍然应该易于阅读。下面列出的一段Sass代码将帮助你解决这个问题。它使用Sass中内建的颜色函数帮你挑选一种与背景色相衬并且对比度更高的文本颜色。下面的代码位于本章代码示例的color-helper.scss文件中，如代码清单10-1所示。

代码清单10-1　有帮助的颜色函数

```
// 如果颜色的明度高于50%则返回true
@function is-bright($color) {
    @return (lightness($color) > 50%);
}

// 如果亮则返回$light值，暗则返回$dark值
@function if-bright($bg, $light: true, $dark: false) {
```

```
    @return if(is-bright($bg), $light, $dark);
}

// 选取对比度最高的颜色
@function text-contrast($bg, $dark-text: #000, $light-text: #fff) {
    @return if-bright($bg, $dark-text, $light-text);
}
```

上面的例子非常好地说明了Sass如何解决在Web设计中遇到的一系列问题，从而改变设计师和开发者使用样式表所能做的事。你在这里可以看到分享Sass和分享CSS之间的不同之处。你也能从中了解到，当样式表作者在分享和重用工具而不是具体的实现方式时，编写样式表最终如何参与到了Web开发的世界中。但是简单地将一个Sass文件或者代码片段发布到网络上是远远不够的。使用Compass扩展，你可以做得更好。

10.1.3 分享Sass是远远不够的

可以看到，Sass比CSS更容易分享，但是当你拥有一段想要分享的Sass代码片段时，最好的分享方式是什么呢？你可以将它发布到博客上，甚至可以将它作为一个迭代的样例发布在CodePen（http://codepen.io）上。其他人可以简单地将你的代码复制到他们的项目中，并开始享受你的劳动成果。但是如果你想要分享的不仅仅是一个代码片段怎么办？

如果你想要在分享Sass代码的同时分享一些其他的资源，例如图片、字体、JavaScript或者HTML，应该怎么办？这就要求用户将这些文件移动到合适的位置以便将你的成果整合到他们的项目中。你分享的内容越精细，用户就要花费更多的精力才能开始使用你分享的内容。

10.1.4 为什么使用Compass扩展

Compass扩展是分享Sass脚本（以及相关联的资源）的一种绝好方式，使他人能轻松地使用你的成果。Compass扩展同时也为个人或者企业级别的框架提供了一个非常好的构建代码块。你应该编写真正的Compass扩展，而不是将Sass代码块从一个项目复制到另一个项目。

当有人安装了你的Compass扩展，你可以肯定他们可以使用由Compass提供的相同混入类库、函数库以及其他特性的库。这使我们可以轻松地依靠Compass包含的库来创建扩展。如果你想要使用精灵图或者CSS3，你可以充满自信地使用Compass中的工具，无需担心会增加扩展的复杂性。

Compass扩展可以包含多个项目模板来向用户展示其不同特性，并为用户提供一个使用示例代码的起点。根据配置文件，Compass知道资源位于什么位置，因此你的模板可以简单地在正确的地方安装样式表、图片或者JavaScript。

上面所讲的内容也许会让你觉得创建一个Compass扩展好像要花很多功夫，但是不要害怕。创建你自己的Compass扩展其实很简单。

10.2 一个简单的扩展

在下面这个例子中，我们将使用前面的颜色函数并将它们打包成一个名为color-helpers的简

单扩展。如果你现在就想看一看最终结果，可以在本章的代码示例中找到这个简单的扩展。目录结构如代码清单10-2所示。

代码清单10-2　最简单的扩展

```
color-helpers/
  stylesheets/
    color-helpers.scss
```

它是一个以color-helpers扩展命名的目录，其中包含一个stylesheets目录和一个Sass文件，这个Sass文件同样以你的扩展命名。这就是一个最基本形式的扩展。你可以将它作为一个Github项目分享，或者将它压缩为zip文件上传到你的网站。这样的形式一般被称为ad hoc扩展。你也可以将扩展以Ruby gem的形式发布，这让用户可以使用命令行下载、安装并升级你的扩展。我们将在稍后探讨这种方式。

10.2.1　安装ad hoc扩展

要安装这种类型的扩展，Compass用户应该将color-helpers目录复制到项目的extensions文件夹中。在扩展被放置到适当的位置之后，这个扩展的Sass文件便可以被引入，好像它们就位于这个项目的sass目录中一样。

对于一个独立的项目，扩展应该放置在位于项目根目录下的extensions目录中，例如project_root/extensions/color-helpers/。在Rails项目中，ad hoc扩展应该被安装到vendor/plugins/compass_extensions目录中。Compass在默认情况下不会自动创建这些目录，因此扩展作者需要手动添加这些目录。

10.2.2　测试你的扩展

为了试用你的新扩展，你需要创建一个新的Compass项目并安装color-helpers扩展。你可以将下面的指示输入终端，也可以在本章代码示例中查看test-color-helpers目录：

```
compass create test-color-helpers --bare
```

现在你需要添加一个sass/screen.scss文件并创建extensions目录。将color-helpers扩展复制到extensions目录中，这时你的项目应该如图10-1所示。

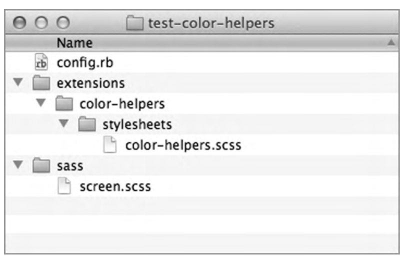


图10-1　color-helpers项目设置

现在你需要添加`@import "color-helpers";`到screen.scss文件中，然后你就可以开始在项目中使用这些颜色函数了。在代码示例中，你将看到几个使用`color-contrast`函数的例子。不可否认，它确实不是一个让人兴奋的扩展。两个颜色函数不会为你提供太大的帮助。正如前面提到的，扩展有时不仅仅是样式表，那就让我们来做一些更有趣的事情：CSS3按钮！

10.3　创建扩展演示项目

正如你所看到的，扩展并不是一个形式完全的Compass项目，而是一个用于Compass项目的文件库。开发扩展的最好方式就是创建一个Compass演示项目。稍后，我们将创建一个高级扩展，但是我们首先需要设置一个演示项目。

演示项目有两个作用：第一，它将帮助你开发并测试扩展；第二，在你完成开发之后，可以使用这个演示作为扩展的项目模板。用户可以轻松地将这个演示安装到自己的Compass项目中，这样可以使用户快捷地尝试你的扩展。

我们这个扩展的目标是帮助用户轻松地生成漂亮的CSS3按钮。我们把这个扩展叫做nice-buttons，有谁会不喜欢漂亮的按钮呢？演示项目的目录结构如图10-2所示。

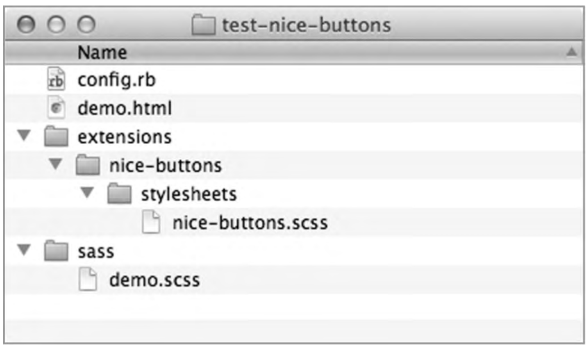


图10-2　nice-buttons演示项目设置

和前面提到的color-helpers扩展一样，你拥有一个基本的Compass项目，它包含一个以你的扩展命名的扩展目录，其中有一个stylesheets目录，该目录中包含一个以你的扩展命名的Sass文件。这里唯一的区别是，你拥有一个demo.html文件，用于在浏览器中预览样式。这个演示HTML不需要很花哨。你只需要在其中放置一个按钮和一个扩展的链接即可，如代码清单10-3所示。

代码清单10-3　demo.html文件

```
<!DOCTYPE html>
<html>
  <head>
    <title>Nice Buttons - Demo</title>
    <link href="stylesheets/demo.css" rel="stylesheet"
                                                type="text/css">
  </head>
```

```
  <body>
    <h1>Button Test</h1>
    <button>Click Me!</button>
    <a class="button" href="#">Click Me!</a>
  </body>
</html>
```

你可能会感到奇怪，为什么我们使用<button>Launch</button>而不是<input type="button" value="Launch">。在过去，input按钮很难在不同浏览器和操作系统中保持同样的样式。在这一点上，现代浏览器更加容易使用，但是HTML 4.01标准（http://www.w3.org/TR/html401/interact/forms.html#h-17.5）指出，创建<button>元素是为了提供更富渲染的可能性。于是，当我们想要在操作系统或者浏览器中使用默认样式时，我们倾向于使用input按钮。当我们想要编写一些自定义样式时，可以依靠<button>元素来保持一致性。

接下来，你需要在demo.scss中引入nice-buttons扩展并添加一段Sass代码，如代码清单10-4所示。

代码清单10-4　demo.scss文件

```
@import 'nice-buttons';
html {
  font-family: Helvetica, Arial, sans;
  background: #f4f4f4;
}
body {
  text-align: center;
  position: absolute;
  top: 30px; left: 30px; bottom: 30px; right: 30px;
  padding-top: 20px;
  background: #fff;
  border: 1px solid #e5e5e5;
}
```

由于你还没有为按钮和链接编写样式，图10-3所展示的就是demo.scss编译后的显示结果。



图10-3　nice-buttons演示——编写样式前

现在你的演示项目已经设置完毕，可以开始编写扩展了。

10.4　编写高级扩展

在我们进入有趣的CSS3部分之前，先来编写一些简单的按钮重置样式是一个不错的想法。根据操作系统和浏览器的不同，按钮有不同的默认样式，由于链接常常和按钮显示为同样的样式，你要确保两者在不同的浏览器和操作系统中都保持一致的样式。因此，你需要在nice-buttons.scss文件中添加nice-button混入类，如代码清单10-5所示。

代码清单10-5　nice-buttons.scss文件

```
@import "compass/css3";

@mixin nice-button() {

      // 重置样式
      font: normal 16px/18px "Lucida Grande", Lucida, Arial, sans-serif;
      margin: 0;
      text-decoration: none;
      cursor: pointer;
      padding: .5em 1.2em;
      @include border-radius(.3em);
      &:active, &:hover { outline: none }

      // 普通样式
      background-color: #eee;
      border: #bbb 1px solid;
      color: #333;
}
```

由于你会依赖Compass的CSS3混入类，你需要确保在文件的顶部引入Compass的CSS3模块。如果用户已经引入了CSS3模块，这个引入语句将会被忽略。现在你只需将这个混入类添加到demo.scss文件中：

```
button, .button { @include nice-button }
```

现在再次打开浏览器，浏览器显示的内容说明你已经走上了正轨（图10-4）。



图10-4　样式相同的按钮

按钮和链接现在看起来完全一样了，但是远远谈不上漂亮。你需要继续修改。

10.4.1　自动化完成困难的部分

你已经完成了基础工作，现在是时候来编写这个扩展的精华部分了。你想让用户可以通过简单地把想要的颜色告诉混入类来创建一个漂亮的按钮。为了自动化完成其中的一些样式，你需要依赖Sass中聪明的颜色转换函数以及本章前面的两个颜色函数。

我们继续在extensions/nice-buttons/stylesheets/中创建_color-helpers.scss文件，在其中添加三个颜色函数，并用`@import "color-helpers";`在nice-buttons.scss的顶部加入该文件。接下来，你需要修改混入类来接受一个背景色并挑选相关颜色，如代码清单10-6所示。

代码清单10-6　nice-buttons.scss

```
@mixin nice-button($bg: #eee) {
    ...
  // 普通样式
  background-color: $bg;
  color: text-contrast($bg, $dark-text: mix($bg, #000, 25%));
  border: scale-color($bg, $lightness: -20%) 1px solid;
}
```

现在，这个混入类让用户设置一个背景色（如果用户不传递任何值则默认为`#eee`），然后，通过使用`text-contrast`函数挑选一个和背景色相衬的文本颜色。将深色文本和背景色稍加混合，以便文本与按钮的其他部分更好地融合。最终，通过加深背景色来选取边框颜色。快速更新demo.scss文件，如代码清单10-7所示。

代码清单10-7　demo.scss

```
button { @include nice-button } //默认的混入类背景
.button { @include nice-button(#494e57) } //深蓝灰色
```

图10-5展示了你的进展。



图10-5　开始自动化颜色选择

这些颜色函数在简化方面做的不错，但是按钮需要一些渐变。为了使这个混入类保持简单，我们在一个另外的“支持”混入类中生成渐变，如代码清单10-8所示。

代码清单10-8 nb-gradient——nice-buttons.scss

```
@mixin nb-gradient($bg) {
  // 测量挑选的主要颜色
  $top:      scale-color($bg, $lightness: 40%);
  $middle-1: scale-color($bg, $lightness: 10%);
  $middle-2: scale-color($bg, $lightness: -5%);
  $bottom:   scale-color($bg, $lightness: -20%);

  @include background-image(linear-gradient(
    $top, $middle-1 50%, $middle-2 50%, $bottom));
}
```

有了这些变量，你可以创建一个带有顶部反光、底部阴影以及模糊中线的漂亮渐变，这使得按钮拥有了三维立体的观感。当创建诸如nb-gradient这样的支持混入类时，最好将它们归入以扩展首字母命名的命名空间中。这样可以使混入类名称保持简短，同时也告诉那些阅读源码的人：这是一个内部混入类，你可能不会使用到它。将@include nb-gradient($bg)添加到nice-button混入类中，你可以得到如图10-6所示的结果。



图10-6 初始渐变样式

深色按钮看起来不错，但是浅色按钮的渐变看起来对比度有些高。为了解决这个问题，我们将区别对待浅色和深色背景，如代码清单10-9所示。

代码清单10-9 nb-gradient混入类中的颜色赋值

```
// 测量挑选的主要颜色
$top:      scale-color($bg, $lightness: if-bright($bg, 80%, 40%));
$middle-1: scale-color($bg, $lightness: if-bright($bg, 20%, 10%));
$middle-2: scale-color($bg, $lightness: if-bright($bg, -2%, -5%));
$bottom:   scale-color($bg, $lightness: if-bright($bg, -10%, -20%));
```

现在你的渐变混入类可以通过调节颜色转变来改进最终的渐变了。if-bright函数将查看

$bg变量：如果$bg亮度高于50%则使用第一个百分比，否则使用第二个百分比。这样的方式比分别编写浅色和深色的渐变代码要简单得多。虽然它只是在百分比上做了微小的调整，但是最终效果却有显著的提升（图10-7）。



图10-7　调节颜色转变后的改进情况

修改前后的差别并不大，但是这种程度的修改却能帮助你创建一个优秀的扩展。当你添加文本阴影和盒阴影时，也可以采用同样的方法来操纵颜色，如代码清单10-10所示。

代码清单10-10　text-shadow——nice.buttons.scss

```
text-shadow: scale-color($bg, $lightness:
    if-bright($bg, 25%, -25%)) 0 1px 1px;
@include box-shadow(rgba(#fff,
    if-bright($bg, .6, .2))    0 0 1px 1px inset);
```

对于文字阴影，if-bright函数决定是否将颜色加深或减淡。对于盒阴影，它选择正确的透明度。你可以看到这个简单的自动化操作有多么强大。图10-8展示了你到目前为止的成果。



图10-8　加入text-shadow并嵌入box-shadow

现在让我们来改善现有的结果并添加一些交互式样式和一个漂亮的CSS3过渡。

编写按钮在聚焦、悬浮和激活状态的样式时，普遍的做法是减淡、加深颜色和添加阴影。编写这种效果的方式有很多种，但是由于你已经开始依赖CSS3，你应该考虑最终生成的CSS文件大小。在本书出版时，你依然需要使用CSS3的厂商前缀，虽然Compass会为你处理好这件事，但是最终输出的文件体积会很大。为悬浮或激活状态生成一个新的亮度或者灰度渐变要在输出中添加很多CSS语句。如果人们使用这个扩展来为许多不同的按钮添加样式，影响将是很显著的。

为了在添加这些互动样式的同时保持输出的精简，你需要使用一个聪明的技巧。当你的文件中包含nb-gradient混入类时，你需要使用一个部分透明色。这意味着按钮的背景色会透过渐变显示出来。现在你可以改变渐变后面的背景色，修改效果将会通过渐变显现出来。下面是到目前为止的主要按钮样式，如代码清单10-11所示。

代码清单10-11　nice-button混入类中的一般按钮样式——nice-buttons.scss

```
// 普通样式
background-color: $bg;
border: scale-color($bg, $lightness: -20%) 1px solid;
color: text-contrast($bg);

@include nb-gradient(rgba($bg, .7)); // alpha显示过渡
@include transition(background-color,box-shadow .15s);
text-shadow: scale-color($bg, $lightness:
   if-bright($bg,25%,-25%)) 0 1px 1px;
@include box-shadow(rgba(#fff,
   if-bright($bg,.6,.2)) 0 0 1px 1px inset);
```

现在我们来添加按钮状态样式，如代码清单10-12所示。

代码清单10-12　添加:hover和:focus的样式——nice-buttons.scss

```
// 状态样式
&:hover, &:focus {
  background-color: scale-color($bg,
    $lightness: if-bright($bg, 85%, 25%)
  );
}

&:active {
  background: scale-color($bg,
    $lightness: if-bright($bg, 55%, 15%)
  );
  border-color: rgba(#000, if-bright($bg, .4, .8));
  @include box-shadow(
    if-bright($bg,
      rgba(mix($bg, #000, 25%), .4),
      rgba(mix($bg, #000), .9)
    ) 0 2px 12px inset
  );
}
```

基本上来说，你只是在按钮被按下时调整了背景色并添加了一个深度嵌入的盒阴影。在激活状态下，你并没有设置background-color，而是设置了background属性，移除渐变背景图片

使按钮只剩下背景色和一个嵌入盒阴影，以此创建了一个阴暗的效果。图10-9逐一展示了三种状态效果。



图10-9　互动按钮状态——nice-buttons.scss

你可能很难通过截图感知到点击按钮的感觉，因此务必要查看示例代码的演示。为这个扩展编写样式的部分已经结束了，但是还可以做一些重构。

10.4.2　重构你的扩展

现在，扩展的主要混入类`nice-buttons`由三个不同部分组成：

(1) 重置样式；

(2) 普通按钮样式；

(3) 按钮状态样式。

重置样式对于每个按钮都相同，每个按钮都使用相同的CSS3过渡。因此只要某文件包含了`nice-button`混入类，就相当于复制了八行Sass代码。现在你还需要为厂商前缀添加额外生成的CSS，这是你现在需要解决的问题。

这是使用`@extend`的最佳场合。你可以将这些样式添加到一个基本类，然后每一个按钮按照如下方式扩展样式：

```
.button-reset {
  // 重置样式
  ...
}
@mixin nice-button {
  @extend .button-reset;
  ...
}
```

但是这都不是你要做的事。

在这种情况下，编写一个扩展和为你自己的项目编写相同的样式存在不同之处。即使.button-reset可能是安全的类名，但是这违反了扩展设计的一个关键原则：除非有要求，否则扩展不应该输出任何CSS代码。扩展中的样式应该尽可能地存在于混入类中，否则，你需要假设用户会使用哪些元素或类名，引入扩展中的样式表可能导致用户设计中的元素自动继承你的样式。

你可以通过将重置样式放在一个混入类中来达到同样的效果，然后将重置混入类包含在一个占位符选择器下：

```
@mixin nice-button-reset() {
  // 重置样式
}
%nice-button-reset { @include nice-button-reset; }
```

正如你在第2章中看到的，如果占位符从未得到过扩展，其中的样式就不会被编译为CSS。这对于扩展作者来说是一件好事，因为它让你的扩展享受选择器占位符继承带来的好处，而无需生成任何不必要的样式或者类名。

为什么我们使用一个重置混入类，而不是仅在占位符下面编写样式？因为一个储存了重置样式的混入类可以在任何地方重用。也许在某个人的项目中，其样式有可能重载了你的重置样式。使用一个混入类，可以轻易地将其包含进样式中来防止层叠。

运用这条原则，扩展的结构概要如代码清单10-13所示。

代码清单10-13 扩展概要

```
@mixin nb-reset() {
  // 重置样式
}
%nb-reset { @include nb-reset }

@mixin nb-gradient($bg) {
  // 渐变样式
}

@mixin nice-button($bg: #eee) {
  @extend %nb-reset
  // 普通样式
  // 按钮状态样式
}
```

将所有的重复移除之后，你的扩展看起来苗条多了。它能生成漂亮的按钮和CSS。我们现在来看一看构成扩展的全部62行Sass代码，如代码清单10-14所示。

代码清单10-14 扩展目录模式

```
@import "compass/css3";
@import "color-helpers";

// 按钮样式重置和基本样式
@mixin nb-reset() {
```

```
  font: normal 16px "Lucida Grande", Lucida, Arial, sans-serif;
  margin: 0;
  text-decoration: none;
  margin-bottom: .3em;
  cursor: pointer;
  padding: .5em 1.2em;
  display: inline-block;
  border: { width: 1px; style: solid }
  @include border-radius(.3em);
  &:active, &:hover { outline: none }
  @include transition(background-color,box-shadow .15s);
}

%nb-reset { @include nb-reset; }

// 使用简单的色移来自动选取渐变
@mixin nb-gradient($bg) {
  $top:      scale-color($bg, $lightness: if-bright($bg,80%,40%));
  $middle-1: scale-color($bg, $lightness: if-bright($bg,20%,10%));
  $middle-2: scale-color($bg, $lightness: if-bright($bg,-2%,-5%));
  $bottom:   scale-color($bg, $lightness: if-bright($bg,-10%,-20%));
  @include background-image(linear-gradient(
    $top, $middle-1 50%, $middle-2 50%, $bottom));
}

@mixin nice-button($bg: #eee) {
  @extend %nb-reset;

  // 一般样式
  background-color: $bg;
  border-color: scale-color($bg, $lightness: -20%);
  color: text-contrast($bg);
  @include nb-gradient(rgba($bg, .7)); // alpha显示过渡

  text-shadow: scale-color($bg, $lightness:
    if-bright($bg,25%,-25%)) 0 1px 1px;
  @include box-shadow(rgba(#fff,
    if-bright($bg,.6,.2)) 0 0 1px 1px inset);

  // 状态样式
  &:hover, &:focus {
    background-color:
                 scale-color($bg, $lightness: if-bright($bg, 85%, 25%));
  }

  &:active {
    background: scale-color($bg,
      $lightness: if-bright($bg, 25%, 10%));
    border-color: rgba(#000, if-bright($bg, .4, .8));
    @include box-shadow(
      if-bright($bg,
        rgba(mix($bg, #000, 25%), .4), rgba(mix($bg, #000), .9)
      ) 0 2px 12px inset
    );
  }
}
```

图10-10展示了一个使用该扩展的例子。



图10-10 nice-buttons颜色试验

现在，使用这个扩展，任何Compass用户都能够使用一行Sass代码创建漂亮的CSS3按钮。但是你现在还有工作要做。是时候学习将这个演示打包成一个示例项目了，然后我们将看看应该如何分享这个扩展。

10.5 创建一个模板

你的扩展已经能够正常运行，是时候准备对外发布了。既然你已经有了一个不错的工作演示，你可以将它纳入扩展来向新用户展示nice-buttons扩展的运行方式。Compass允许扩展作者加入模板来帮助用户快速开始使用扩展。对于你创建的这种扩展，模板仅仅是一个演示；但是对于更大型、更复杂的扩展，你可能需要使用模板来引导启动一个复杂的框架。下面展示了一个使用模板和Sass扩展的扩展目录结构，如代码清单10-15所示。

代码清单10-15 扩展目录模式

```
my-extension/
  stylesheets/
    my-extension.scss
  templates/
    project/
      manifest.rb
      test.html
      test.scss
  lib
    my-extension.rb
    my-extension/
      sass_extensions.rb
```

你没有必要对nice-buttons扩展编写一个Sass扩展，但是将你的演示作为一个可安装的模板会很好。为了将你的演示转换成一个模板，你需要复制demo.html和demo.scss到nice-buttons/templates/project中。然后你需要创建一个`manifest.rb`文件，它将帮助Compass定位和识别资源。

`manifest.rb`在templates/project目录下，将模板需要的资源编制成列表。你也可以添加项目描述、帮助文本（运行`compass help nice-buttons`时显示），以及一段在安装扩展时显示的欢迎信息。下面展示的是为nice-buttons编写的`manifest.rb`文件，如代码清单10-16所示。

代码清单10-16　`nice-buttons/templates/project/manifest.rb`

```
stylesheet 'demo.scss', :media => 'screen, projection'
html 'demo.html'
image 'screenshot.png'

description "Create beautiful CSS3 buttons from a single color"
help "This will install a demo (HTML and Scss) to show you how to use
  nice-buttons"
welcome_message %Q{
Example usage: button { @include nice-buttons(#444) }
See demo.html and demo.scss for example usage.
See screenshot.png for a screenshot of the rendered demo.
Enjoy!
}
```

在扩展中包含文档，使用欢迎信息和帮助文本支持URL（如果你的扩展含有URL）也许都是非常好的主意。

将演示作为模板添加进入扩展之后，扩展的目录结构如代码清单10-17所示。

代码清单10-17　nice-buttons目录结构

```
nice-buttons/
  stylesheets/
   _color-helpers.scss
    nice-buttons.scss
  templates/
    project/
      demo.html
      demo.scss
      manifest.rb
```

如果用户在他们项目的extensions目录中包含了nice-buttons扩展，他们可以通过运行下面的命令来安装默认模板：

```
compass install nice-buttons
```

这条命令将使用项目的Compass配置把templates/project中的资源复制到合适的位置。如果你在templates/warm-cookies目录中创建了第二个模板，用户就可以通过传递目录名称来安装它：

```
compass install nice-buttons/warm-cookies
```

在发布之前，你最好再为扩展添加一个README文件来描述扩展的作用、使用方法和安装方法。但是鉴于本章编写的目的，对创建扩展的讲述就到此为止了。你的扩展现在拥有一个不错的演示，并且已经做好了分享的准备。接下来，我们将看看如何发布这个扩展以便其他人使用。

10.6　分发扩展

过去，Web设计社区中主要的分享方式是博客，使用的是zip压缩文件。尽管开发者喜欢通过复杂的版本控制和分发渠道来发布软件，但设计师则依赖于更简单的方法。如今，越来越多的设计师参与到了开源之中，是时候学习如何正确地分发软件了。

这是一个非常深入的话题，因此本节内容只介绍不同分发方法的概念，不打算详细讲述分发开源软件的方法。

10.6.1　在存档中分发扩展

分发Compass扩展的最简单方法是将它压缩为一个zip文件并上传到存档中，然后将它放置到某个服务器上。尽管这种方法不占用时间，但是有几个缺点。

首先，更新需要用户自己手动替换旧的代码。当文件位于多个地方时，这将会是一件非常痛苦的事。其次，由于没有严格的版本控制，当你必须在一个项目的几个不同历史版本下工作时，试图维护旧的发布版本是一个巨大的挑战。尽管版本控制系统会保存每个文件的所有历史版本并提供标签之类的功能，但是存档仅仅是你的扩展在某个时间点的快照。这会带来诸多的局限。

10.6.2　将扩展作为Ruby gem分发

Ruby gem是打包和分发代码的一种精密方式。针对我们的需求，发布gem的最有利之处是内建的依赖管理。你可以要求扩展的用户具有其他特定版本的gem，如Sass和Compass。当然，你对于任何分发都可以添加一个README文件来说明最低支持版本，但是对于gem，只是简单地安装gem就将同时获取并安装你所依赖全部gem的正确版本。

使用一个ad hoc扩展，用户必须将扩展中的所有代码复制到他们项目的extensions目录中。但是使用gem，扩展代码将存在于一个集中的位置，这允许多个项目使用同样的扩展代码。

1. 将ad hoc扩展转换为gem

为了将你的扩展作为Ruby gem分发，你需要在你的项目中添加几个文件，如图10-11所示。

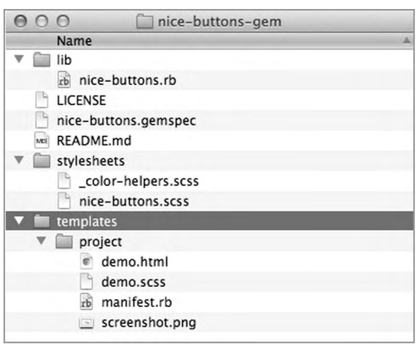


图10-11　gem项目设置

一个gem至少需要拥有一个gemspec和一个lib目录,目录包含一个以该gem命名的Ruby文件。你将使用nice-buttons.rb文件在Compass中注册你的扩展，并告诉Compass扩展目录的位置。下一个代码清单展示了nice-buttons.rb的代码，如代码清单10-18所示。

代码清单10-18　nice-buttons.rb

```
require 'compass'

Compass::Frameworks.register('nice-buttons',
  :stylesheets_directory => File.join(File.dirname(__FILE__), '..',
     'stylesheets'),
  :templates_directory => File.join(File.dirname(__FILE__), '..',
     'templates'))
```

上面的代码可能看起来不如你过去看到的Sass代码漂亮,但是基本上来说,你只是在告诉gem它需要Compass gem。然后，使用一个Compass函数来注册一个叫做nice-buttons的扩展并告诉Compass可以在哪里找到stylesheets和templates目录。

现在，让我们来看看nice-buttons.gemspec文件。组织一个gemspec有很多不同的方式，但是我们只看其中一种简单的方法，如代码清单10-19所示。

代码清单10-19　nice-buttons.gemspec

```
# -*- encoding: utf-8 -*-
Gem::Specification.new do |gem|
    gem.name          = "nice-buttons"
    gem.version       = "1.0.0"
    gem.authors       = ["Brandon Mathis"]
    gem.email         = ["brandon@imathis.com"]
    gem.description   =
                     "Easily create beautiful CSS3 buttons with Compass."
    gem.summary       = "Nice and easy CSS3 buttons for Compass users"
    gem.homepage      = "http://github.com/imathis/nice-buttons"

    gem.files         = %w(README.md LICENSE)
    gem.files        += Dir.glob("lib/**/*")
    gem.files        += Dir.glob("stylesheets/**/*")
    gem.files        += Dir.glob("templates/**/*")

    gem.add_dependency  "sass", ">= 3.2"
    gem.add_dependency  "compass", ">= 0.12"
end
```

这个文件有两个部分。第一个部分由关于gem的元数据组成：它的名字、版本号、作者等。第二部分包括gem的文件，并设置对其他gem的依赖。由于你的扩展依赖于Sass和Compass较新的特性，要求用户拥有的版本不低于Sass 3.2和Compass 0.12。

经过以上这些设置，你可以通过下面的这条命令让Ruby gems生成你的gem文件：

```
gem build nice-buttons.gemspec
```

这条命令将在项目的根目录中生成一个叫做nice-buttons-1.0.0.gem的gem文件。

2. 发布gem

为了发布这个gem，你需要在RubyGems.org（一个用来托管gem的免费中央仓库）创建一个账户，并根据步骤进行设置。当一切就绪，你可以通过运行下面这条命令来发布你的gem：

```
gem push nice-buttons-1.0.0.gem
```

运行之后，任何人都可以立即安装你的gem并开始使用你的扩展。

3. 安装gem

你可以通过在终端中运行下面这条命令来手动安装你的gem：

```
gem install nice-buttons
```

这条命令将从RubyGems.org获取gem，如果你没有合适版本的Compass或Sass，它会为你自动安装。很明显这比使用一个zip压缩文件要好。

为了开始在一个Compass项目中使用你的gem，你需要在Compass配置文件的顶部添加下面一行代码：

```
require "nice-buttons"
```

现在Compass了解了你的gem，你可以引入nice-buttons样式表并在你的项目中使用它。如果你喜欢，也可以通过下面的命令行来安装你在前面创建的演示项目：

```
compass install nice-buttons
```

这条命令将解压你的演示文件到Compass项目，并重新编译你的样式表。过去，这是安装gem并在Compass中使用的最普遍方法，而最近，许多开发者已经开始使用一种叫做Bundler的技术来使用和管理gem。

4. 使用bundler安装gem

Bundler（http://gembundler.com）是一个帮助你在项目中安装并管理Ruby gem的gem。Bundler使用一个Gemfile文件列出项目中正在使用的gem。下面展示的是如何将你的gem添加到列表中，如代码清单10-20所示。

代码清单10-20 在Gemfile中添加nice-buttons

```
source :rubygems

group :assets do
  gem 'nice-buttons'
end
```

Gemfile更新完毕之后，你可以通过一条简单的命令来安装gem：

```
bundle
```

这条命令将连接到RubyGems.org，找到nice-buttons gem的最新版本以及所有依赖，并在你的系统中进行安装。它同时创建了一个叫做Gemfile.lock的文件，其中包含你的项目中正在使用的gem完全列表，及其版本号、下载源和gem依赖的层级关系。这些详细信息被保存下来，以确保不会因为使用了不兼容的版本而出现问题。手动完成这些工作将会非常痛苦，因此Bundler的普

及并不令人意外。

为了和Compass一起使用Bundler，在你的Compass命令前面加上`bundle exec`，如下所示：

```
bundle exec compass compile
```

这条命令将使Compass获取Gemfile中的资源组，它在那里能找到你的扩展并使其为你的项目所用。如果想要安装你创建的演示项目，你可以运行下面这条命令：

```
bundle exec compass install nice-buttons
```

在你的gem安装完成之后，用户可以引入你的nice-buttons样式表并开始使用你的代码。

将扩展作为一个Ruby gem分发非常容易，比起在存档中分享扩展，它带来的好处完全值得我们付出努力。但是在你和别人分享代码之后，他们很有可能会发现bug，想要提供一些改善建议。这又带来了一个新的挑战，即与其他开发者交流并讨论代码贡献，尤其是在你的扩展变得受欢迎之后。为了解决这个问题，我们将介绍Github，这是一个与开源社区进行合作的优秀资源。

10.6.3　在Github上进行代码社交

现在有很多开源社区，但是Github是到目前为止最受欢迎的。Github将在一个精心设计的网站上托管你的代码，方便其他人浏览并下载。它也能为你提供工具来管理贡献者，发布项目网站，并且编辑wiki。其他的Github用户可以复制你的项目，做出改进，然后将他们做出的修改反馈给你。使用Github的issue tracker，你可以回顾并接收代码贡献，将它们直接从网站合并入你的项目。Github向所有开源项目免费提供上述服务。

为了将你的扩展发布在Github上，你需要添加一个新的仓库，并依照简单的指令来提交并推送你的扩展。如果将一个README文件添加到你的项目，Github将会在你的项目主页中展示它。你现在就可以在Github中查看nice-buttons gem的主页http://github.com/imathis/nice-buttons，如图10-12所示。

如果你发布了一个ad hoc扩展，用户可以从项目网站下载nice-buttons，或者通过在extensions目录中运行下面这条命令来安装它：

```
git clone https://github.com/github_user_name/nice-buttons.git
```

之后，用户可以通过在他们的extensions/nice-buttons目录中运行下面的命令来更新到最新版本：

```
git pull
```

这条命令将下载扩展的最新版本，但是如果你的扩展需要一个更新版本的Sass、Compass或者其他gem，用户需要手动将其更新到正确的版本。Github是一个托管开源代码的好地方，你可以在此与他人合作，但是如果你想要依赖管理，就应该将你的代码作为一个Ruby gem来分发。

关于如何分发和管理开源软件，还有很多需要学习，但是本节提供了一个良好的起点。记住一条通用的准则：尝试是最好的学习方式。

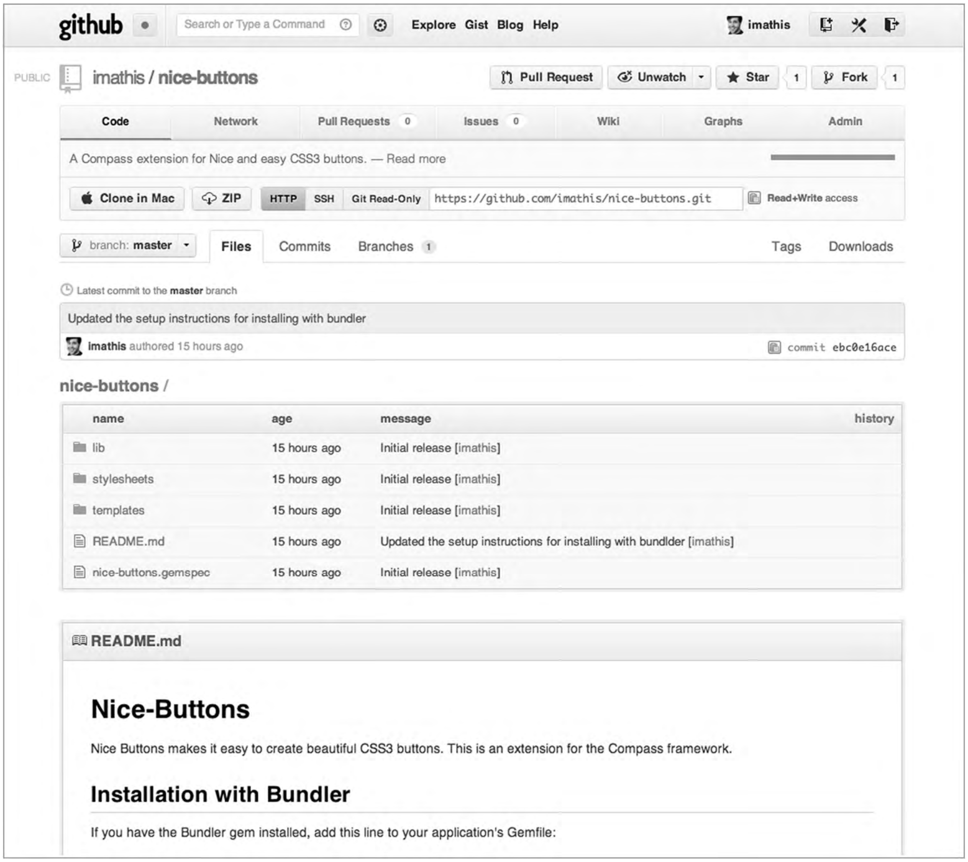


图10-12　nice-buttons项目页面

10.7　小结

在本章中，你了解到Sass和Compass不仅能提高你解决样式表问题的能力，同时也让你以一种从未有过的方式来分享知识和经验，参与到设计社区中。我们探索了如何编写旨在与他人分享的样式表。我们在一步步创建、重构以及打包一个优秀的Compass扩展的过程中探讨了设计优秀扩展的原则。我们也了解了分发扩展的不同方法，并学习了分享代码以及同其他人合作的方式。

通过对本书全部内容的学习，你已经了解了Sass作为一门动态样式表语言的威力，以及它怎样帮你编写更易读、更具可维护性的样式表。你了解了Compass如何提供一个可靠的库，顺畅地整合到你的开发环境中，并为你提供一个构建与分享知识的绝佳平台。通过为你提供新颖有趣的挑战和更好的回报，Sass和Compass击败了无聊的CSS。无论你是Sass和Compass新手还是老手，我们都希望本书能够让你以全新的视角来看待样式表，掌握新的技巧，加深对样式表的理解，并能够大胆地自由发挥，尝试更好的创意。

附录 A

安装Sass和Compass

Sass和Compass都是基于Ruby编程语言的命令行工具。要使用它们，你首先需要在电脑中安装Ruby，并对电脑的命令行操作有一个基本的理解。Sass和Compass可以安装在Windows、Mac OS X和Linux系统中。

A.1 在Windows系统中安装

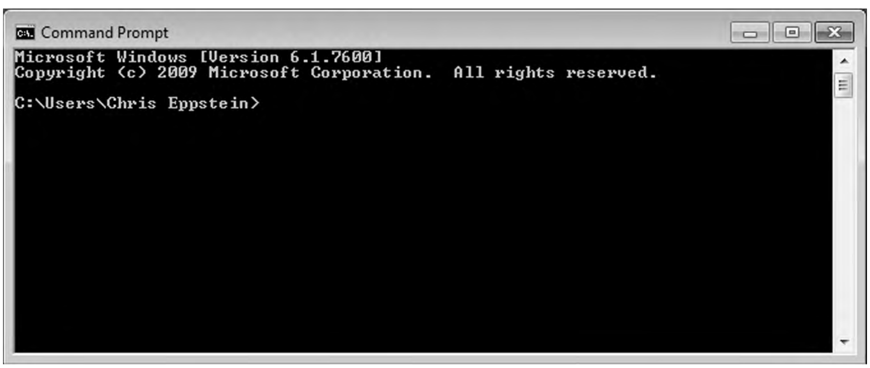
Windows系统并没有预置Ruby，因此如果你之前没有安装过Ruby，现在就需要进行安装。安装Ruby只需要花费几分钟的时间。

A.1.1 打开Windows命令行窗口

在Windows 7系统下，你可以在Window开始菜单中选择 程序 > 附件 > 命令行 来打开命令行窗口。你也可以在搜索框中输入`command`，并从搜索结果中选择命令行。

在Windows 7之前的Windows版本中，你可以通过选择 属性 > 附件 > 命令行 来打开命令行窗口。你也可以选择 运行，然后输入`cmd`并按下回车键。

当命令行开始运行时，你应该看到像图A-1这样的窗口。



图A-1　Windows命令行窗口

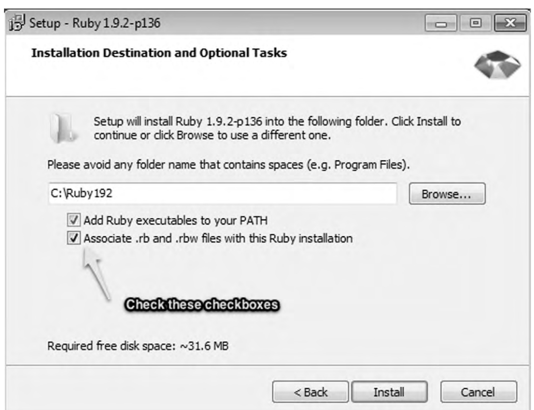
A.1.2　在Windows中安装Ruby

在命令行窗口中输入`ruby -v`并按下回车键。如果电脑中没有安装Ruby，命令行将显示`'ruby' is not recognized as an internal or external command, operable program or batch file`。如果电脑中已经安装了Ruby，那么将打印出已经安装好的Ruby版本号。你安装的Ruby版本应该不低于1.8.7（如果你安装的Ruby版本为1.8.6或者更低，请按照下面的指示重新安装Ruby，如图A-2所示）。



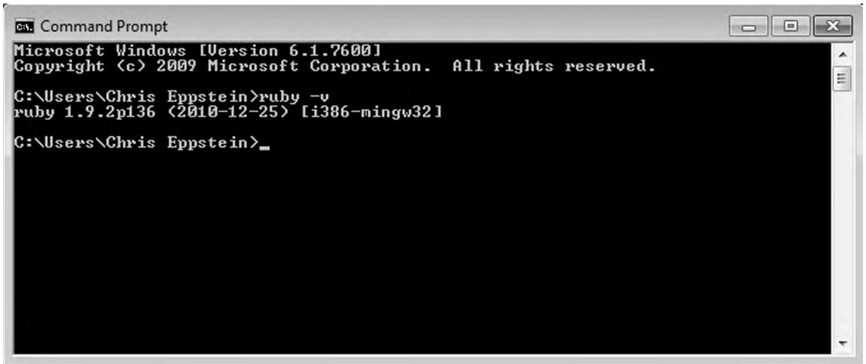
图A-2　下载Ruby安装程序

进入网址http://rubyinstaller.org/downloads/，下载最新版本的Ruby安装程序并运行。根据安装程序的提示按照步骤进行安装。第三个窗口将询问你将Ruby安装在什么位置，在点击安装之前选中两个复选框，如图A-3所示。



图A-3　配置Ruby安装程序

现在，重新启动你的命令行窗口，然后输入`ruby -v`并按下回车键来确认Ruby已经安装完成，如图A-4所示。



图A-4　Ruby安装完成

A.1.3　在Windows系统中安装Sass和Compass

Ruby自带一个叫做RubyGems的系统，用来安装基于Ruby的软件。我们可以使用这个系统来轻松地安装Sass和Compass。要安装最新版本的Sass和Compass，你需要输入下面的命令：

```
$ gem install sass
$ gem install compass
```

在每一个安装过程中，你都会看到如下输出：

```
Successfully installed sass-3.1.0
1 gem installed
Installing ri documentation for sass-3.1.0...
Installing RDoc documentation for sass-3.1.0...
```

安装完成之后，你应该通过运行下面的命令来确认应用已经正确地安装到了电脑中：

```
$ sass -v
Sass 3.1.0 (XXX NAME ME)

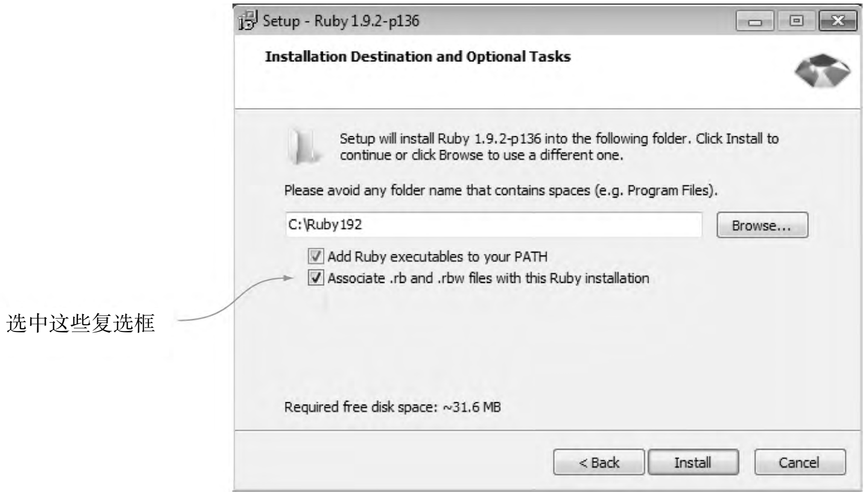
$ compass -v
Compass 0.11.0
Copyright (c) 2008-2011 Chris Eppstein
Released under the MIT License.
```

A.2　在MAC OS X系统中安装

一段时间以来，MAC OS X系统中都默认安装有Ruby，因此你可能并不需要进行安装。

A.2.1 打开Mac OS X终端窗口

你可以在Finder中通过点击 应用程序 > 实用工具，然后双击终端.app来运行终端应用程序，如图A-5所示。



图A-5　运行Mac OS X终端

如果你不熟悉Mac OS X终端，我们向你推荐John Long编写的终端教程，参见http://wiseheartdesign.com/articles/2010/11/12/the-designers-guide-to-the-osx-command-prompt/。

A.2.2 安装Ruby

在命令行中输入`ruby -v`并按下回车键。如果你的电脑中没有安装Ruby，命令行将显示`bash: ruby: command not found`。这种情况不太可能发生，如果发生，请按照http://rubyosx.rubyforge.org/的安装步骤安装Ruby，并确保在安装完成之后重启终端窗口。

A.2.3 在Mac中安装Sass和Compass

Ruby自带一个叫做RubyGems的系统，用来安装基于Ruby的软件。我们可以使用这个系统来轻松地安装Sass和Compass。要安装最新版本的Sass和Compass，你需要输入下面的命令：

```
$ sudo gem install sass
$ sudo gem install compass
```

在每一个安装过程中，你都会看到如下输出：

```
Successfully installed sass-3.1.0
1 gem installed
Installing ri documentation for sass-3.1.0...
Installing RDoc documentation for sass-3.1.0...
```

安装完成之后，你应该通过运行下面的命令来确认应用已经正确地安装到了电脑中：

```
$ sass -v
Sass 3.1.0 (XXX NAME ME)

$ compass -v
Compass 0.11.0
Copyright (c) 2008-2011 Chris Eppstein
Released under the MIT License.
```

A.3　在Linux系统中安装

如果你的Linux系统中没有安装Ruby，请参照Linux系统发布版本的指示来安装Ruby。

A.3.1　打开Linux终端

默认Linux用户知道如何打开终端。

A.3.2　安装Ruby

如果你的电脑中没有安装Ruby，使用Linux发布版本的软件安装机制来安装Ruby。

A.3.3　在Linux系统中安装Sass和Compass

Ruby自带一个叫做RubyGems的系统，用来安装基于Ruby的软件。我们可以使用这个系统来轻松地安装Sass和Compass。要安装最新版本的Sass和Compass，你需要输入下面的命令：

```
$ sudo gem install sass
$ sudo gem install compass
```

在每一个安装过程中，你都会看到如下输出：

```
Successfully installed sass-3.1.0
1 gem installed
Installing ri documentation for sass-3.1.0...
Installing RDoc documentation for sass-3.1.0...
```

安装完成之后，你应该通过运行下面的命令来确认应用已经正确地安装到了电脑中：


```
$ sass -v
Sass 3.1.0 (XXX NAME ME)

$ compass -v
Compass 0.11.0
Copyright (c) 2008-2011 Chris Eppstein
Released under the MIT License.
```

附录 B 开始使用Compass

B.1 创建一个新项目

要在一个新项目中使用Compass，打开你的终端窗口并运行下面的命令：

```
$ compass create my-project
```

如果my-project目录不存在，上述命令会创建一个叫做my-project的目录并在其中添加以下文件：

```
my-project/
  config.rb
  - sass/
    - ie.scss
    - print.scss
    - screen.scss
  - stylesheets/
    - ie.css
    - print.css
    - screen.css
```

如果你没有为`compass create`命令传递一个目录参数，它将使用你当前所在的目录。

在`config.rb`文件中，你可以对Compass的一些配置进行修改，例如资源位置和压缩程度（我们将在后面深入讨论）。sass目录包含了一些初始的样式表，你可以对它们进行编辑、重命名，或者完全抛弃，但是这个文件夹是你存放Sass样式表的地方。最后，stylesheets目录用于存放编译后的CSS文件。

B.1.1 在设置项目时配置选项

使用`compass create`命令时，你可以使用一些选项来配置你的项目：

```
--bare                  (不包含默认样式表进行安装)；
--syntax sass           (在默认样式表中使用缩进语法)；
--sass-dir "cool"       (使用'cool'目录存放Sass文件)；
--css-dir "style"       (使用'style'目录存放CSS文件)；
--images-dir "img"      (使用'img'目录存放图片)；
--fonts-dir "type"      (使用'type'目录存放字体文件)；
--javascripts-dir "js"  (使用'js'目录存放JavaScript文件)。
```

添加选项后的命令如下：

```
$ compass create my-project --bare --sass-dir "cool" --css-dir "style"
```

你也许会感到疑惑，为什么可以设置一个JavaScript目录？这是因为Compass扩展还可以打包相关联的JavaScript文件，这样的设置可以让你在安装扩展时告诉Compass应该在什么位置存放这些JavaScript文件。

B.1.2　在Rails项目中添加Compass

为了在一个Rails项目中安装Compass，首先要使用`cd`命令进入你的项目目录，并在Gemfile文件中添加以下内容：

```
group :assets do
  gem 'compass-rails'
  # Add any compass extensions here
end
```

然后在终端中运行下面的命令：

```
$ bundle
$ bundle exec compass init rails
```

在Rails项目中，Compass配置文件位于`config/compass.rb`。

如果你的项目位于Rails 2.3或者3.0，你需要参照`compass-rails`README文件中列出的步骤，参见https://github.com/Compass/compass-rails/blob/master/README.md。

B.2　安装Compass扩展

Compass扩展以两种形式发布：Ruby gem和ad hoc扩展。两种形式的扩展都非常容易安装并且能在所有项目中良好地协同工作。如果想了解如何开发扩展，可以阅读第10章相关内容。

B.2.1　安装以Ruby gem形式发布的扩展

你可以将扩展安装到你的系统gem中，如下所示：

```
$ sudo gem install extension-name
```

如果你使用Bundler，将下面的内容添加到你的Gemfile中：

```
gem 'extension-name'
```

然后在终端中安装gem：

```
$ bundle install
```

现在你已经将gem下载到了系统中，接下来需要将它安装到你的项目中。

B.2.2　为已经存在的项目安装扩展

既然你已经拥有了一个以gem形式发布的扩展，就可以将以下内容添加到config.rb文件来通知Compass：

```
require 'extension-name'
```

然后，根据你的项目目录，在终端中运行下面命令：

```
$ compass install -r extension-name -f extension-name
```

现在你可以开始在项目中使用这个扩展了。你可以使用compass install命令同时安装多个扩展，只要确保命令行中的每个扩展前都使用了-r和-f标示符。

为新项目安装扩展

如果你已经安装了扩展的gem，可以使用这个扩展创建一个新项目。运行下面的命令：

```
$ compass create my-project -r extension-name --using extension-name
```

这将创建一个新的Compass项目，并对其进行配置以运用你的扩展。

B.2.3　安装以ad hoc形式发布的扩展

Ad hoc扩展只是一些目录，其中包含Sass样式表以及若干向Compass说明样式表运行方式的文件。如果你的项目不包含一个extensions目录，创建一个然后将扩展的文件夹复制进去。你的项目文件结构应该如下所示：

```
my-project/
  config.rb
  extensions/
    some-extension/
  sass/
  stylesheets/
```

为了在一个Rails app中安装ad hoc扩展，你需要在vendor/plugins/compass_extensions目录下创建一个extensions目录。

你可以通过在项目的配置文件中设置extensions_dir来自定义扩展所在的目录。

B.2.4　安装扩展的模式

大多数扩展都自带一个默认的模式，其中包含一个样式表或者一个与该扩展一起使用的文件。这些模式会连同compass install脚本一起自动安装。但是一些扩展作者会创建额外的模式，这些模式提供使用例子，或者用初始的文件或者样板代码助你一臂之力。安装一个扩展模式的命令如下所示：

```
$ compass install extension-name/pattern-name
```

Compass将会显示扩展作者包含在其中的指令（如果有的话）以及一个随模式安装的新文件列表。

解压扩展和框架

对一些读者来说，使用作为Ruby gem安装的扩展可能有些不同寻常。有些时候阅读一个扩展的源码也是很有帮助的，但是Ruby gem的源码可能储存在电脑的别处。为了解决这个问题，Compass提供了一项功能，可以将扩展、甚至Compass框架本身解压到你的项目目录中。你需要用到下面的命令：

```
$ compass unpack extension-name
$ compass unpack compass
```

该命令会将与扩展和Compass框架相关的文件解压到项目的extensions目录中。因此现在你的项目看起来如下：

```
your-project/
  extensions/
    compass-13.0/
    extension-name-1.0/
```

Compass同时会输出一个友好的警告，本质意思是“只可远观而不可亵玩”。它可能会试图改变你刚才解压的代码，但修改代码是一个不好的举动，将导致你在升级扩展时失去自定义配置。这项功能的最佳用途是教学或者排除错误。

B.3 配置你的Compass项目

Compass是一个Sass库，一个扩展的平台，也是一个用于整合项目环境的系统。Compass配置将这些部分整合起来，让你的工作流程更加顺畅，更加灵活。

B.3.1 使用资源

在编写样式表之外，样式表作者还会频繁地使用图片、字体和JavaScript，这些文件经常是相互联系的。例如，为了显示一张背景图片，样式表需要将图片的精确位置告诉浏览器。如果你曾重新组织过一个项目或者改变过一个目录的名字，你一定知道更新这些URL的痛苦。

Compass通过为你编写资源URL，致力于帮助你让一切保持同步。在你的配置文件中，你可以告诉Compass项目资源的位置以及想要生成怎样的URL。如果Compass在你编译样式表时找不到某些资源，它甚至会输出一个警告。

B.3.2 配置资源位置

为了告诉Compass能在文件系统的什么地方找到你的资源，你需要设置以下配置：

- `images_dir`——默认为<project>/images；
- `sass_dir`——默认为<project>/sass；
- `css_dir`——默认为<project>/stylesheets；
- `fonts_dir`——默认为<project>/<css_dir>/fonts；

- javascripts_dir——默认为<project>/javascripts。

这些配置与你的项目目录相关联，因此设置images_dir = img将会告诉Compass在your-project/img/ 目录下查找你的项目图片。然后，在你的样式表中，你可以使用image-url()辅助器来引用一张图片：

```
#logo { background: url('/img/logo.png') }
```

Compass会在your-project/img/logo.png中寻找你的图片，然后对下面的CSS代码进行编译：

```
#logo { background: url('/img/logo.png') }
```

如果要将你写的项目部署到Web服务器的一个子目录中，你可以通过设置http_path配置来自定义URL。你也能为CSS、图片、JavaScript以及字体设置URL配置。

如果你想知道Compass中配置的值，可以使用下面的命令向Compass询问：

```
$ compass config -p sass_dir
app/stylesheets
$ compass config -p css_dir
public/stylesheets
```

想要深入了解Compass配置的有关内容，请参见第8章。

B.4　命令行

Compass中主要的命令有：

- compass create——创建一个新的Compass项目；
- compass init——为一个已经存在的项目（Rails）添加compass；
- compass clean——移除生成的文件和缓存；
- compass compile——生成你的样式表；
- compass watch——监视你的Sass文件并根据变化重新生成文件。

还有一些有用的命令：

- compass stats——查看样式表的统计数据；
- compass unpack <extension>——解压扩展到你的项目；
- compass validate——验证你生成的CSS文件；
- compass version——显示版本、许可证，等等；
- compass interactive——进入一个用于测试Compass中SassScript的控制台。

获得帮助

使用Compass时需要记住许多东西，但是你也可以通过询问命令行来寻求帮助。运行compass help将列出以下信息：

- 带有描述的命令；

- 可用的框架和扩展；
- compass命令的全局选项。

你也可以像下面这样，获取一个子命令的详细帮助信息：

```
$ compass help watch
```

运行这条命令，你将获取关于compass watch命令用法的详细描述，它的语法，以及一列带有详细描述的选项。

你也可以获取关于扩展或者扩展模式的帮助信息：

```
$ compass help extension-name
$ compass help extension-name/pattern
```

如果扩展的作者没有在扩展中添加帮助文本,以上的命令将会显示默认的Compass帮助内容。

附录 C

Sass语法

C.1 缩进Sass与SCSS

本书的大部分章节都在讲述SCSS语法，SCSS代表Sassy CSS。SCSS是CSS的超集，换句话说，任何有效的CSS文件也是一个有效的SCSS文件。这意味着你可以将CSS文件screen.css变为screen.scss，无需做任何其他改变就可以为其添加Sass特性。因此，尽管SCSS语法出现较晚，但它已经变成了最受欢迎的Sass语法。

起初，Sass只有一种语法，叫做*缩进语法*，那时就简称为Sass。为了阐述这两种语法之间的差别，我们将查看使用两种语法编写的相同代码。

首先，我们来看SCSS语法，如代码清单C-1所示。

代码清单C-1　SCSS语法示例

```
/* Compass将CSS3变得容易了!
   尤其是CSS3渐变*/

@import "compass/css3"

// 这个混入类让我们可以轻松地制造渐变
// 它会为我们挑选颜色，很好。

@mixin easy-gradient($bg, $alpha: false) {
  @if ($alpha) {
    $bg: rgba($bg, $alpha);
  }
  $top: lighten($bg, 5);
  $bottom: darken($bg, 5);
  @include background-image(
    linear-gradient($top, $bottom)
  )
}

nav {
  margin: 20px { left: 0; right: 0 }
  @include easy-gradient(#ccc);
  a { color: blue; text-decoration: none }
}
```

现在，让我们用缩进语法编写相同的代码，如代码清单C-2所示。

代码清单C-2　缩进语法示例

```
/* Compass将CSS3变得容易了！
   尤其是CSS3渐变*/

@import compass/css3

// 这个混入类让我们可以轻松地制造渐变
// 它会为我们挑选颜色，很好。

=easy-gradient($bg, $alpha: false)
  @if ($alpha)
    $bg: rgba($bg, $alpha)
  $top: lighten($bg, 5)
  $bottom: darken($bg, 5)
  +background-image(linear-gradient($top, $bottom))

nav
  margin: 20px
    left: 0
    right: 0
  +easy-gradient(#ccc)
  a
    color: blue
    text-decoration: none
```

上述两段代码有一些明显的不同，还有一些微妙的差别。让我们来一一解析。

C.1.1　空格与括号、分号

最显著的差别是，缩进语法没有花括号和分号。在SCSS语法使用常见花括号的地方，缩进语法使用空格。同样，缩进语法使用新行来分隔属性，不使用分号。

那些喜欢使用缩进语法的人称，缩进语法移除了起干扰作用的符号，使他们的Sass更加简洁，易于阅读。那些喜欢SCSS的人享受随意使用空格的自由，将多个属性放在一行或者将一个长函数分割为多行。他们也喜欢无需移除字符，也无需按照缩进语法严格的空格要求重新转换格式，就能够使用标准CSS语法。

除了使用空格或字符这个最明显的差别，两者还有其他的不同之处。

C.1.2　`@import`指令

在SCSS中，`@import`指令需要用括号将目标包裹起来，但是缩进语法则不需要括号。还有非常重要的一点是，使用`@import`指令时，文件扩展名不是必须的。你可以使用文件扩展名，但是在不使用的情况下，你可以引入.scss文件或者.sass文件。使用指令`@import "some-file";`，Sass将会查找some-file.scss文件和some-file.scss文件。因此，两种语法可以轻松共存，你可以在使

用SCSS语法编写样式表的同时，引入他人使用缩进语法编写的扩展。

C.1.3　混入类

创建并使用混入类的方法也有不同之处。在SCSS中，`@minxin`以及`@include`指令分别用来定义并使用混入类：

```
@mixin easy-gradient($bg, $alpha: false) { ... }
@include easy-gradient(#ccc);
```

缩进语法其实可以使用与SCSS相同的指令，也可以使用`=`来代替`@mixin`，使用`+`来代替`@include`：

```
=easy-gradient($bg, $alpha: false)
+easy-gradient(#ccc)
```

尽管缩进语法的用户指出，其混入类指令十分简洁，但还是有些人喜欢编写明显的`@minxin`和`@include`，这也是它们在缩进语法中依然可以使用的原因。

C.1.4　注释

在Sass中，有三种不同的注释。以`//`开头的注释不会出现在生成的CSS中；以`/*`开始的注释将会出现在未经压缩的CSS中；以`/*!`开头的注释在压缩和未经压缩的CSS文件中都会出现。在缩进语法中，如果作者将每行缩进在注释标记下方，所有的注释都会变成多行注释，如下所示：

```
// some comment
   which spans
   multiple lines

/* This comment
   spans multiple
   lines too

/*! As does
    this one
```

在SCSS中，`//`注释是单行注释，其余两个多行注释必须包含在注释符号内，如下所示：

```
// some comment
// which spans
// multiple lines

/* This comment
   spans multiple
   lines too */

/*! As does
    this one !*/
```

C.1.5　哪种语法更好

Sass爱好者围绕这个话题展开了激烈的争论，双方各执一词，也有人承认会根据不同目的使用两种语法。幸运的是，你并不需要做出抉择。Sass项目的维护人员已经决定保留两种语法，你也可以使用`sass-convert`命令行工具在两种语法之间进行转换。由于`@import`指令的灵活性，你可以轻松地在同一个项目中使用两种语法，但是不能在同一个文件中同时使用两种语法。

译者介绍



刘炬光

腾讯Web前端工程师，AlloyTeam核心成员。热爱HTML5 WebApp开发，在客户端结合WebApp的开发上有非常丰富的经验。注重开发流程自动化，主导所在团队的Web开发自动化工作。他是国内最早接触Sass和Compass的程序员，曾在w3ctech等大会上向大家介绍Sass和Compass。新浪微博：@materliu。



赵锦江

山西太原人士。2007年本科毕业于西北工业大学软件工程专业，并从事前端开发工作至今，现任淘宝（中国）软件有限公司前端开发专家，花名“勾股”。爱好翻译，长期参与W3C HTML IG（中文兴趣组）的技术规范翻译工作。新浪微博：@勾三股四。



张浩然

本科专业为计算数学，现在南开大学攻读计算数学硕士。目前专注于前端开发，持续关注最新的前端技术。爱前端，爱翻译，曾在“前端乱炖”网站翻译过70余篇文章，网站ID“张小俊128”。新浪微博：@张小俊128。



Sass and Compass IN ACTION

Sass与Compass实战

“本书可以让网页设计师和开发人员从重复劳动中解放出来，体会撰写CSS的乐趣。”

——薛良斌，Compass.app & Fire.app开发者

“我对本书的出版充满期待，并相信它一定会把更多人带入CSS的世界。”

——张克军，豆瓣前端负责人

“Sass与Compass创建者提供的最重要的一手资料，必读！”

——David A. Mosher， DAVEMO Consulting

“本书作者就是那些将CSS转换成了更有趣内容的人。”

——Jeroen van Dijk，ADGOJI

“解决现实世界问题的优秀示例。”

——Jacob Rohde，Amino

还在用CSS手工编写网页布局？还在用“查找/替换”修改十六进制颜色值？随着项目规模日益增长，组织样式表会令你抓狂。你没有想到，自己把大部分时间都浪费在了这些细枝末节上，哪里还有时间去创新？Sass和Compass将你从这些重复工作中解放出来，引入变量、混合器、选择器嵌套、选择器继承等概念，为标准CSS添加了脚本功能和组件库，帮助Web设计师和开发人员大幅度简化了样式表编辑工作。它们能自动完成很多枯燥乏味的任务，并在页面中添加动态样式，进而快速精准地呈现网页。

本书是使用这两种革命性工具创作样式表的权威指南，内容循序渐进，示例结合实战。四位作者都是Sass和Compass的核心团队成员（及创建者），向读者展示了高效使用这两种框架的精髓。

本书适合所有Web设计师、前端开发人员、Web应用产品经理及相关人员阅读。

MANNING
图灵社区：iTuring.cn
热线：(010)51095186转600
分类建议 计算机/Web开发
人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-35301-6
ISBN 978-7-115-35301-6
定价：49.00元